Java™ magazine

By and for the Java community

//JULY/AUGUST 2012 /

DEVELOPER POWER

With a bevy of cloud-based tools available, it's a great time to be a Java developer 34

ORACLE.COM/JAVAMAGAZINE

ORACLE

//table of contents /

PHOTOGRAPH BY BOB ADLER

//from the editor /

Java developers are known to be passionate, even religious, about their IDEs and other tools. With the emergence of a whole slate of new open source and commercial tools (virtually all of them cloud-based) over the past five years, there's an awful lot to be passionate about.

There are several reasons for this Cambrian explosion of solutions in this area. For one, agile development practices, which are now extremely common on a global scale, cry out for collaborative tools that make the continuous build process manageable. Development teams are also more geographically dispersed than ever before. Furthermore, the popularity and easy availability of many thousands of open source modules and frameworks, which are otherwise often integrated into an application's code base without much regard for their provenance, make code governance a bit more important than in the past.

There is also a cloud-related factor to be considered on the supply side of the equation: cloud computing has simply made it possible for many of these solutions to be brought to a broad market in a relatively frictionless way. Furthermore, online communities like Java.net and GitHub have wrapped up some of these technologies in nice social packages that speed adoption through network effects.

But enough abstract discussion. Jump to this issue's special feature on the topic, "Developer Power" (not intended to be exhaustive, just representative).

Speaking of passion: Zoran Severac's "Java People" performance impressed us so much that we're opening a general Java Nation call for songs. If you're a songwriter who appreciates Java and the Java community, send us a link to your composition. It may become our official Java Nation anthem!

Justin Kestelyn, Editor in Chief BIO

PHOTOGRAPH BY BOB ADLER

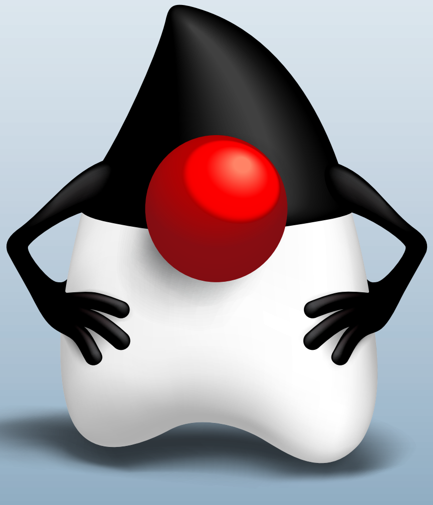
//send us your feedback /

We'll review all suggestions for future improvements. Depending on volume, some messages may not get a direct reply.

Java™

GIVE BACK!
ADOPT A JSR

Find your JSR here



Safari® Books Online

Answers. Fast.

Subscribe and gain instant cutting-edge development insight from the world's most trusted tech publishers — all in one on-demand digital library.

- Get unlimited access to 20,000+ online books and training videos from100+ publishers
- Search full text to efficiently get the answers you need
- Make notes and organize content into folders you control
- Elevate your career with thousands of top business and professional development titles

ANYTIME, ANYWHERE:
Access Safari Books Online on virtually any device with a browser.

Start a free unlimited-access trial today safaribooksonline.com/javamag

GET READY FOR JAVAONE 2012

JavaOne 2012 comes to San Francisco, California, September 30–October 4. It's time to make your plans, because you won't want to miss this conference. JavaOne—known for its educational offerings, community, and networking—is a place to improve your working knowledge and coding expertise, learn from the world's foremost Java experts, meet with fellow developers in formal and casual settings, and enjoy one of the world's great cities.

This year's JavaOne promises a wealth of content focused on the evolution and future of Java technology as well as tools, resources, and best practices that developers need to design and build rich, compelling, and individualized services across diverse technologies. Conference planning is still under way, but here are the highlights:

PHOTOGRAPH BY SAUL LEWIS

The Zone will be the hub of JavaOne activity September 30–October 4.

Keynotes. The conference kicks off on Sunday, September 30, with the Java Strategy, Partner, and Technical keynotes beginning at 4 p.m. To accommodate the ever-growing number of attendees, keynotes will be held at the historic Masonic Auditorium on Nob Hill. After the keynotes, attendees can head to the official JavaOne Open House at the Taylor Street Café in the Zone.

Sessions. Sessions start on Monday, October 1, and attendees can choose from hundreds of expert-led technical sessions, hands-on labs, panel discussions, and birds-of-a-feather sessions. Check out the JavaOne 2012 track titles and descriptions.

Exhibition Hall. The Java Exhibition Hall brings innovation and practical learning into one convenient location: the Zone, which features exhibitor booths, technology demonstrations, partner exhibits, and more. Don't miss the chance to network with Java experts, developers, Oracle and its partners, and architects from around the world.

Rock stars. This year's conference will again feature JavaOne "rock star" speakers, recognized for both their Java technical expertise and their speaking skills in the event's highest-rated sessions.

Java University. The Java University preconference gives you a chance to get even more from your conference experience by attending a full day of in-depth Java training delivered by experts on Sunday, before the conference officially kicks off with the opening keynote at 4 p.m.

Register. Make your travel and lodging plans, and register now!

JavaOne Russia 2012

Attendance topped 2,000 at JavaOne Russia, April 17–18, in Moscow—clearly showing that interest in Java technologies in Russia is on the rise. At the Core Java Platform keynote, Oracle's **Georges Saab** (above right) and **Mike Lehmann** (above left) spoke to a packed house. JavaOne Russia featured more than half of its sessions in Russian, and Java booths were staffed by at least two engineers. A highlight included the Hackzone, where participants could bring their issues and get suggestions immediately.

Left to right: JavaOne India attendees learn hands on; Oracle's Nandini Ramani provides a technology roadmap; attendees discuss what sessions to attend.

JAVAONE INDIA 2012

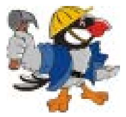
More than 2,000 people attended JavaOne India, held May 3–4 in Hyderabad. This regional JavaOne conference included informative keynote and technical sessions, hands-on labs, and an Oracle Technology Network room for community-related presentations.

The Java Strategy keynote reminded the Indian Java community of the power and strength of Java and reaffirmed Oracle's commitment to Java. Oracle Java engineering executives **Nandini Ramani** and **Anil Gaur** provided an overview of the technology roadmap for Java, including plans for Java SE 8, the focus on the cloud for Java EE 7, and the Java SE and ME/embedded convergence. "Oracle has aggressive plans for Java over the next few years, and we are continuing to drive technical advancements across the platform," Gaur said.

At the Nokia keynote, **Gerard J. Rego**, head of ecosystem and developer experience at Nokia, provided an overview of the state of mobile technology. The keynote included a video about a cool mobile application called Nano Ganesh that controls water pumps via mobile phone, a boon for Indian farmers so they don't have to walk to their pumps at night. The developer, **Santosh Ostwal**, came on stage to discuss his application and encouraged developers to use their creativity to solve problems. "Don't think of it as a phone; think of it as a low-cost wireless device," he said. See Nano Ganesh in action.

In the technical sessions and at the booths, there was lots of interest in JavaFX, Java EE 6, JDK 8/9/10, and Project Avatar. In addition, many Java user group (JUG) leaders from across India came to JavaOne India to network with community members and share their expertise. Watch them talk about their JUGs and the challenges they face.

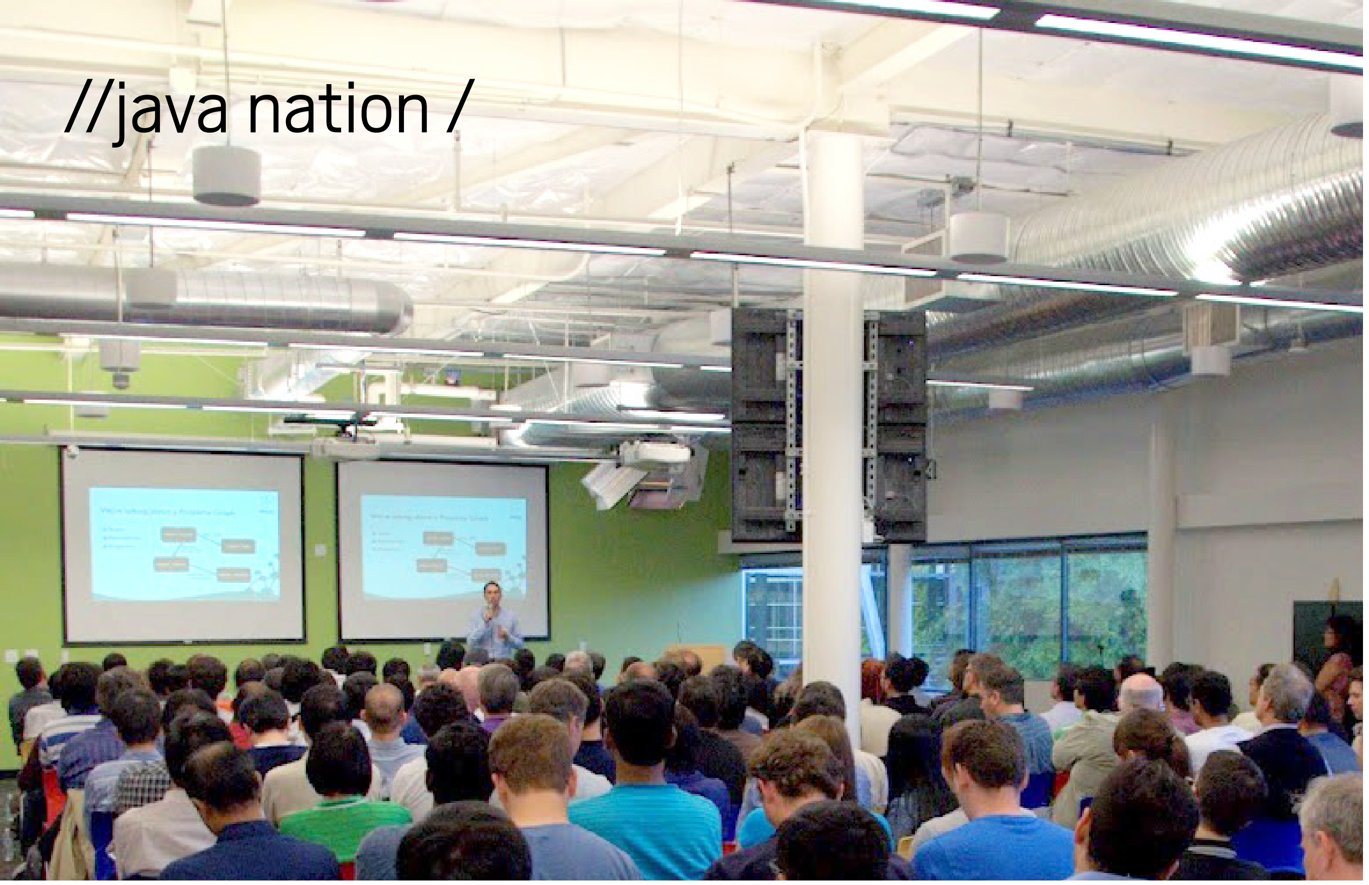
Java.net's Java Tools Community

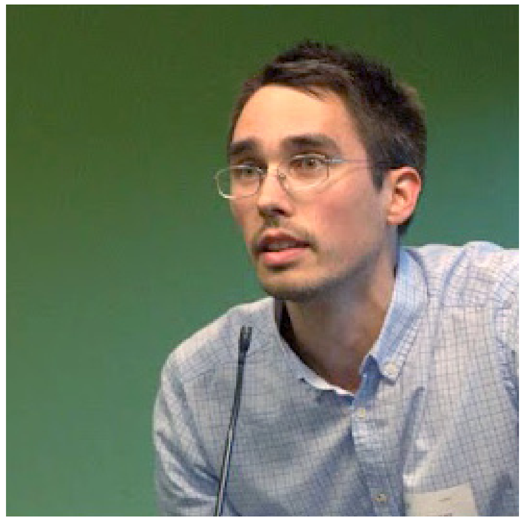


The Java Tools Community is one of the most active communities on Java.net. Led by **Fabiane Bizinella Nardon**, **Anton (Toni) Epple**, and **Daniel López Janáriz**, the community supports the development of open source Java development tools on Java.net, offering guidance and feedback to Java Tools project leaders and visibility to their projects.

Key projects associated with the Java Tools Community include VisualVM (a tool that visually integrates command-line JDK tools and lightweight profiling capabilities); Hudson CI (a continuous integration [CI] tool); Jailer (a tool that provides data exportation of referentially intact row sets); ThreadLogic (a graphical interface for analyzing Java thread dumps); Java Application Bundler (a tool that packages a Java application as a Mac application bundle); and JTHarness (a fully featured test framework that facilitates comprehensive unit testing).

If you have an idea for an open source Java tool, consider hosting your project on Java.net and joining the Java Tools Community.





At a Silicon Valley JUG meeting in May, speaker David Montag of Neo Technology provided a high-level introduction to graph databases.

JAVA USER GROUP PROFILE

Silicon Valley JUG Mountain View, California

WHEN THE SILICON VALLEY WEB DEVELOPER JAVA USER GROUP WAS FORMED in April 2003 by **Van Riper**, its initial focus was on the Jakarta Struts framework. In July 2004, the focus was broadened to include J2EE. Today, the group covers all Java technologies and as of April 2012 is known as the Silicon Valley Java User Group (SVJUG).

Kevin Nilson joined Riper as coleader in 2007, when the group still had fewer than 100 members. Joining Meetup in 2009 proved to be a turning point in growth. By August 2009, the group had 536 members, which doubled to 1,071 by May 2010. Current membership exceeds 2,300.

SVJUG meets on the third Wednesday of each month at Google headquarters in Mountain View, California, with attendance ranging from 100 to 200 people. The JUG's location provides many advantages, Nilson says, especially with regard to speakers. "Some of the top speakers we have had are **James Gosling**, **Joshua Bloch**, **Rod Johnson**, **Neal Gafter**, **Bob Lee**, **Gavin King**, **Chet Haase**, **Romain Guy**, and **the Java Posse**."

Even the attendees have been known to give presentations in a pinch. "Once I made a scheduling mistake and realized that our speaker was not scheduled to come until one month later," Nilson says. "Luckily jQuery committers **Jonathan Sharp** and **Mike Holster** were in the front row. At 6:55 p.m., I approached them about giving a talk at 7:00 p.m. Jonathan and Mike were happy to fill in and gave a great presentation."

Marakana TechTV records many SVJUG talks, and when the JUG cohosts a meeting with the Silicon Valley JavaFX Meetup, Marakana live-streams the event. "Our group has been very lucky because others have stepped up to help record our talks," notes Nilson. "We do not do any of our own recording, so usually only popular talks get recorded."

SVJUG members also actively participate in the Silicon Valley Code Camp, a community event where developers learn from fellow developers. All are welcome to attend and speak—more than 2,000 developers attended last year. SVCC 2012 will be held October 6–7 (following JavaOne).

PHOTOGRAPHS COURTESY OF SILICON VALLEY JUG

JDK FOR MAC OS X AVAILABLE

Oracle has released its first Java Development Kit (JDK) and JavaFX Software Development Kit (SDK) for Mac OS X. Java developers can now download Java Platform, Standard Edition 7, Update 4 (Java SE 7 Update 4) and JavaFX SDK 2.1 for Mac OS X from Oracle Technology Network. The Java SE 7 Update 4 SDK includes the next-generation Garbage Collection algorithm, Garbage First (G1), which has been highly anticipated by the Java developer community.

JavaFX 2.1 introduces playback support for digital media stored in the MPEG-4 multimedia container format containing H.264/AVC video and Advanced Audio Coding audio. It also includes WebView support for JavaScript to Java method calls. "We look forward to delivering simultaneous releases of the JRE [Java runtime environment] across all major operating systems later this year, so all Java users will be able to take advantage of the latest features and security fixes," says **Hasan Rizvi**, senior vice president of Oracle Fusion Middleware and Java products at Oracle.

ART BY I-HUA CHEN

JAVA AT MAKER FAIRE

Java came to life at the Maker Faire Bay Area 2012 (May 19–20 in San Mateo, California). In the Java Zone in the Expo Hall, attendees explored storyboarding, 3-D animation building, device programming, body motion sensors, game making, and more. They learned how Java interacts with other devices using a motion sensor and a Webcam, and how Microsoft Kinect records motion to animate a Java game. Attendees also played Breakout using the PicoBoard, an external circuit board with sensors for sound and light, slider and button controllers, and four resistance sensors. They also had the opportunity to try out Alice, a tool used to teach Java programming fundamentals using 3-D graphics and a drag-and-drop interface; Greenfoot, a 2-D tool ideal for teaching Java programming basics in high schools and universities; and other tools for the twenty-first-century classroom.

The 2012 Maker Faire also offered several interesting presentations at center stage, including "Gamification, Robotics, and Simulators: How to Get Started with Java Programming." This panel, targeted at students and newbies, focused on why Java technology dominates cutting-edge software development, and provided ways to get started learning programming and Java. *Java Magazine*'s **Justin Kestelyn** moderated the panel of Oracle employees: **Caron Newman**, senior curriculum manager for Oracle Academy; **Daniel Green**, systems engineer; **Kevin Roebuck**, solution specialist with Oracle's global education team; and **Ultan O'Broin**, director of global user experience. They discussed why Java is such a great tool for education.

PHOTOGRAPHS BY SAUL LEWIS

Future Java developers played games at the Java booth at Maker Faire Bay Area 2012.

Oracle CEO Larry Ellison discussed the ease of developing applications for the cloud.

Develop in the Cloud

At an Oracle cloud strategy Webcast on June 6, Oracle CEO Larry Ellison introduced Oracle Developer Cloud Services, a component of Oracle Cloud Platform Services. Oracle Developer Cloud Services will provide instant access to tools that enable faster, smarter, and more-collaborative development in the cloud. With Developer Services, development teams can use their favorite tools (Hudson for continuous integration, Git and GitHub for source control, wiki and tasks for project management) in the Oracle Cloud. Integration with popular IDEs such as Oracle JDeveloper, Eclipse, and NetBeans makes Developer Services an excellent way to maximize productivity and innovation.

JavaSpotlight Podcast

Listen to the JavaSpotlight podcast for interviews, news, and insight for and from Java developers. Hosted by **Roger Brinkley** and **Terrence Barr**, this weekly show includes a rotating panel of all-star Java developers.

FEATURED JAVA.NET PROJECT

JITSI

Blue Jimp developers collaborate using Jitsi, an open source Java VoIP client.

Over its nine-year history, Jitsi has evolved from a Session Initiation Protocol (SIP) research project into a widely used open source Java Voice over IP (VoIP) and instant messaging client that supports five full-time developers working at Blue Jimp. According to Jitsi Project Lead **Emil Ivov**, SIP Communicator (Jitsi's predecessor), released in 2003, did SIP calls, including video. In 2006, additional developers joined the project. The entire code base was reworked, and Jitsi was born. In 2009, Ivov founded Blue Jimp, which offers customers professional support, maintenance, and custom development related to Jitsi.

Several development milestones have increased Jitsi's user base over the years, including the expansion of Jitsi's video footprint from 160 x 120 pixels to 640 x 480 pixels; support for ZRTP and call encryption; and support for Extensible Messaging and Presence Protocol (XMPP) (enabling video calls using Google Talk, Gmail, and Android phones).

The latest Jitsi releases have added the capability to establish conference calls in situations where people are using different protocols; Domain Name System Security (DNSSEC) support; and additional hot-plug support for audio devices.

The development team is currently working on features to be released in the coming months, including the Jitsi Android port and high-quality video conferencing.

More than 100 developers have contributed to the Jitsi code base. Currently, more than 20 developers contribute new code and patches, and many users test snapshot versions and report the issues they find. According to Ohloh, the Jitsi code base consists of more than 700,000 lines of source. About 50,000 users download the application each month. The total user base for Jitsi (including all of its branded incarnations) could approach 1 million people.

"FOSS [free and open source software] is a viable option for anyone setting up a new project," Ivov says. "Business models that rely on selling licenses to end users simply don't scale. With FOSS you get to concentrate on development, and you also gain valuable outside contributions."

This article is the first in a series of Java .net project profiles. Want to have your project profiled in Java Magazine? *Tell us why it's great.*

JAVA CHAMPION PROFILE
JORGE VARGAS

Java Champion Jorge Vargas, freelance Java consultant, blogger, and cofounder and CTO of the geolocalization-based application Yumbling, talked about what he does when he's not working.

Java Magazine: Where did you grow up?

Vargas: In Mexico City. Now I'm living near there in a small town named Calimaya.

Java Magazine: When did you first become interested in computers?

Vargas: When I was 16, my friend Adolfo was very interested in computers, so I started studying and experimenting, too. My first formal project was a Lotus 1-2-3 application I wrote for a building company that calculated project costs. I wrote 4,000 lines of code and was paid about US$300.

Java Magazine: What was your first computer and programming language?

Vargas: The first language I learned was Basic, along with macros in Lotus 1-2-3. Next, I spent some years with Visual Basic (3.x and 4.x). My first computer was a 286 running MS-DOS 4.0, with a 100 MB hard drive and 640 KB of RAM—a real monster!

Java Magazine: What do you enjoy for fun and relaxation?

Vargas: Well, I enjoy programming for fun! I also play piano and enjoy looking at the stars. In school, I spent many fun months building my own telescope.

Java Magazine: What's a typical "family day"?

Vargas: During the work week we talk, and I help with homework if I'm not too busy. We enjoy going to the cinema and eating out. I visit my parents to talk about plants, flowers, or politics, or to play cards or dominoes.

Java Magazine: What side effects of your career do you enjoy the most?

Vargas: I remember a delicious dinner in San Francisco with other Java Champions; it was wonderful because we shared different points of view about our careers. I'm thankful that my career has permitted me to travel, and especially to have close contact with universities and their students.

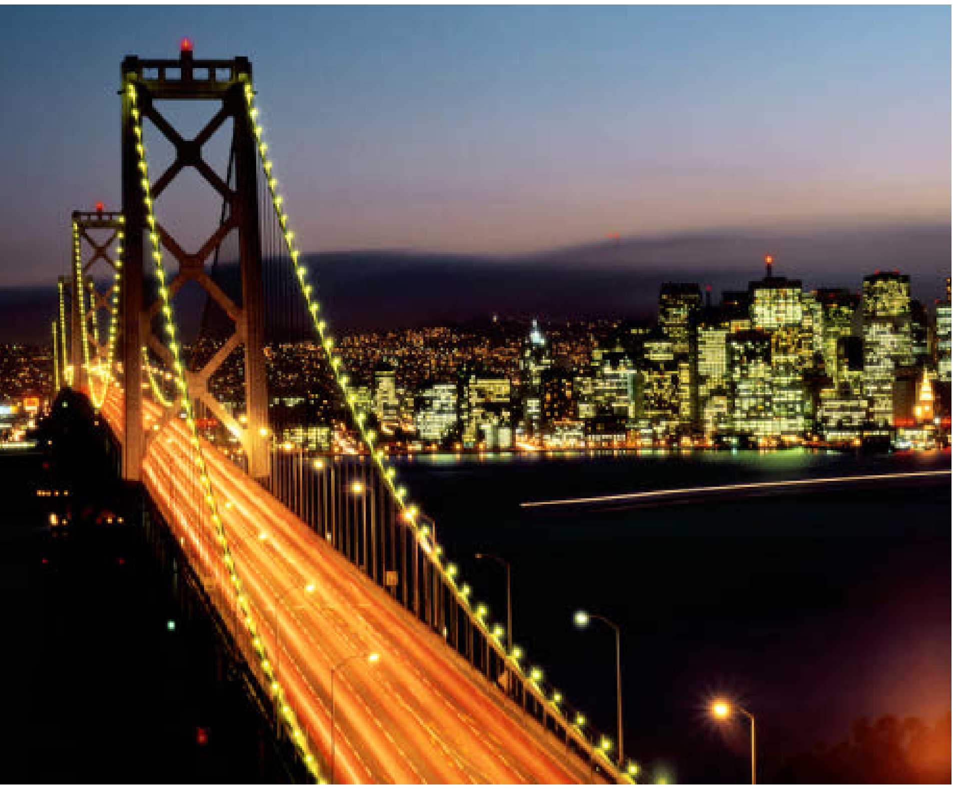
Java Magazine: Has being a Java Champion changed anything for you with respect to your daily life?

Vargas: When I became a Java Champion, I felt I needed to expend more effort sharing my experiences and knowledge at universities, conferences, and workshops. I consider this a responsibility of the role, because people ask me more questions, and value my judgment.

Java Magazine: What are you looking forward to in the coming years?

Vargas: I want to expand my career possibilities and also study piano. Professionally, I'll continue working on Yumbling. And I want to write for *Java Magazine*!

Read more about Jorge Vargas in "An Interview with Java Champion Jorge Vargas." You can also visit his blog and find him on Twitter (@edivargas) and Facebook.



EVENTS

JavaOne 2012 *SEPTEMBER 30–OCTOBER 4, SAN FRANCISCO, CALIFORNIA*

Java continues to move forward. Experts from the worldwide Java community will share unique and leading-edge content with attendees at JavaOne 2012. Tracks range from the stronger-than-ever core Java platform and advancements in tools and techniques that make it easier to write better code in less time to the application development languages that utilize the power of the Java Virtual Machine, including Groovy, JavaScript, JRuby, Kotlin, and Scala. Don't miss it!

AUGUST

QCon
AUGUST 4–5
SÃO PAULO, BRAZIL
QCon is an enterprise software development conference designed for team leads, architects, and project management and is organized by and for the community. Topics include architecture, functional languages, mobile, agile engineering practices, and more. The keynote lineup includes Martin Fowler, chief scientist at ThoughtWorks, and Zach Holman, architect of GitHub.

The No Fluff Just Stuff Software Symposium Tour
Since 2001, the No Fluff Just Stuff (NFJS) Software Symposium Tour has delivered more than 275 events with more than 40,000 attendees. NFJS is known for knowledgeable speakers and timely presentations that cover the latest trends within the Java ecosystem and agility space. Upcoming symposium dates include the following:

Research Triangle Software Symposium
AUGUST 24–26
RALEIGH, NORTH CAROLINA

Pacific Northwest Software Symposium
SEPTEMBER 7–9
SEATTLE, WASHINGTON

New England Software Symposium
SEPTEMBER 14–16
BOSTON, MASSACHUSETTS

Greater Atlanta Software Symposium
SEPTEMBER 21–23
ATLANTA, GEORGIA

SEPTEMBER

Web Developer Conference (WDC) 2012
SEPTEMBER 17–18
HAMBURG, GERMANY
This conference for developers of Web applications, content and online managers, agencies, and Webmasters focuses on professional development of Web applications.

onGameStart 2012
SEPTEMBER 19–21
WARSAW, POLAND
At the self-dubbed "first HTML5 game conference," speakers include Andres Pagella, Jerome Etienne, Jon Howard, Jonas Wagner, Jordan Mechner, and many others.

The Developer's Conference 2012
SEPTEMBER 28–29
FLORIANÓPOLIS, BRAZIL
One of Brazil's largest developer conferences, with locations in São Paulo, Florianópolis, and Goiânia, The Developer's Conference offers dozens of tracks, several of which are Java related, to more than 6,000 attendees.

JAVA OLYMPICS FINAL ROUND RESULTS

The 2011–2012 Java Olympics wrapped up on April 28 with a dramatic final round at Kazakh-British Technical University in Almaty, Kazakhstan. Out of 1,573 students from 322 universities who participated in Round One, only 21 advanced to Round Three, the finals. Only three of these would take home a prize.

After a grueling four hours of testing in which the contestants had to solve five problems, **Igor Ignatiev** (above left), a student at Chelyabinsk State University in Russia, won first place. He had correctly solved four of the five questions in the first 90 minutes of the competition and appeared to have the contest locked up. But he struggled with the last question while the rest of the field advanced on him. In the end, Ignatiev prevailed. **Peter Pchelko** and **Andrey Misnik** finished second and third, respectively.

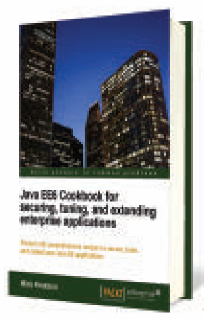
SINGING JAVA'S PRAISES

The community spirit **Zoran Severac** experienced at JavaOne 2011 was the inspiration behind his smash hit (more than 5,000 views on YouTube) "Java People." The song itself is free and open source. "You can listen to it, play it, modify it, and use it for commercial and noncommercial purposes for free as long as you put some reference to the original song," says Severac.

"Java People" and its community concept impressed so many of us here at *Java Magazine* that we're opening a general Java Nation call for songs. If you're a songwriter who, like Severac, appreciates Java and the Java community, start composing! Send us links to your Java Nation anthem.

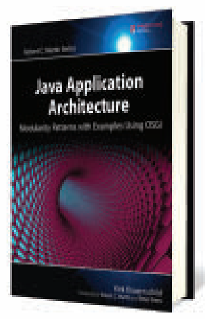
Watch Zoran Severac perform his ode to the Java community, "Java People."

JAVA BOOKS



JAVA EE 6 COOKBOOK FOR SECURING, TUNING, AND EXTENDING ENTERPRISE APPLICATIONS
By Mick Knutson
Packt Publishing (May 2012)

Java EE is a widely used platform for enterprise server programming in the Java programming language, and this book covers recipes on securing, tuning, and extending enterprise applications using a Java EE 6 implementation. It begins with the essential changes in Java EE 6 and dives into the implementation of some of the new features of the Java Persistence API (JPA) 2.0 specification. There are several additional sections that describe some of the subtle issues encountered, tips, and extension points for starting your own JPA application, or extending an existing application.

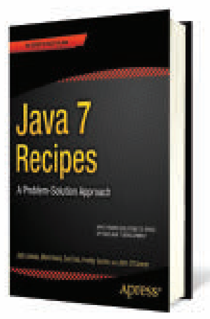


JAVA APPLICATION ARCHITECTURE: MODULARITY PATTERNS WITH EXAMPLES USING OSGI
By Kirk Knoernschild
Prentice Hall Professional (March 2012)

Over the past several years, module frameworks have been gaining traction on the Java platform. *Java Application Architecture* lays the foundation you'll need to incorporate modular design thinking into your development initiatives. Before he walks you through 18 patterns that will help you architect modular software, author Kirk Knoernschild lays a solid foundation that shows you why modularity is a critical weapon in your arsenal of design tools. Throughout, you'll find examples that illustrate the concepts.

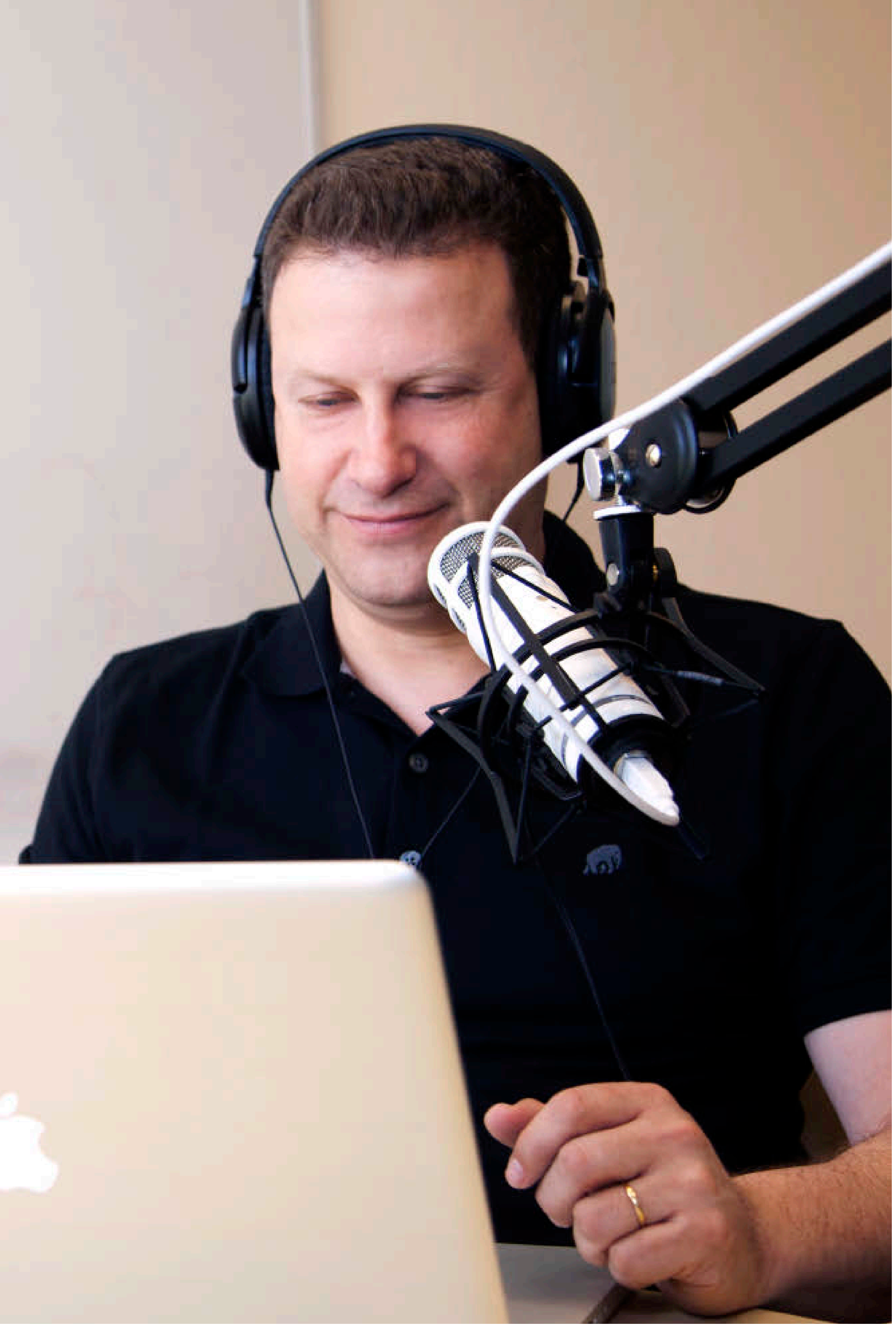
BEGINNING JAVA 7
By Jeff Friesen
Apress (November 2011)

Jeff Friesen's *Beginning Java 7* gets you coding with the new Java 7, Oracle's latest release of the popular Java language and platform. It is the definitive guide to the Java language and the numerous APIs that you'll need to master to become an accomplished Java developer. The book begins with an introduction to Java and focuses on fundamental concepts of the Java language such as comments, identifiers, variables, expressions, and statements. Other topics in the book include classes and objects, advanced language features, language APIs, collecting objects, and much more.



JAVA 7 RECIPES: A PROBLEM-SOLUTION APPROACH
By Josh Juneau, Carl Dea, Freddy Guime, and John O'Conner
Apress (January 2012)

This book offers solutions to common programming problems you may have encountered while developing Java-based applications. Updated to cover the newest features and techniques, *Java 7 Recipes* provides code examples involving Servlets, JavaFX 2.0, XML, Swing, and more. This book uses the popular problem-solution format: you can look up the specific programming problem you want to solve, read the solution, and apply that solution directly in your own code. The authors focus on problems and solutions rather than on the Java programming language itself.



JCP Executive Series

A Conversation with Gil Tene

Azul Systems CTO **Gil Tene** discusses the state of Java and the JCP.

BY JANICE J. HEISS

In the first of a series of interviews with distinguished members of the Executive Committee of the Java Community Process (JCP), we sat down with Azul Systems CTO and Cofounder Gil Tene, who represents Azul on the JCP, to get his take on how things are going with Java and the JCP. Azul Systems, a builder of highly scalable Java Virtual Machines (JVMs), has been a JCP member since 2003 and a member of the Executive Committee since November 2011.

Gil Tene, CTO of Azul Systems, prepares for a Webcast at company headquarters in Sunnyvale, California.

PHOTOGRAPHY BY MARGOT HARTFORD

Java Magazine: Tell us a little about your day job.

Tene: I'm the CTO of Azul Systems, and at Azul we focus on building highly scalable JVMs. So our world is Java and only Java, which is why the JCP and the Java community are central to everything we do. Over the years, we've delivered massively scalable JVMs that can run on platforms with anywhere from a couple to almost a thousand CPU cores, and a handful to hundreds and thousands of gigabytes of memory. We're probably best known for our very concurrent garbage collection and the effective elimination of garbage collection pauses as an issue for response-time-sensitive enterprise Java applications.

I mix both hands-on engineering, where I work within and with the teams at Azul, and a lot of external activity involving conferences and papers and customer interaction. I've been doing this at Azul for about 10 years, and it's the most fun that I've ever had at a job. Azul is like a toy store for engineers. I've specifically worked on various parts of scalable JVMs, including everything from concurrent garbage collectors to locking and transactional memory. We've always had the approach of doing "whatever it takes" for runtime scalability. I've built kernel code and advanced virtual memory management code, designed new processor features and instructions, and played a lot with virtualization.

For the last couple of years, Azul has focused more on software-centric products, so the future trajectory of the Java platform has become increasingly more important to us. We want to contribute to the leadership of the JCP, rather than just being followers. We've been moving closer and closer to the leading edge with Java. As our software is now consumed by more and more people, developers want to interact with it more. While we support a lot of Java EE activity, our technical focus is on Java SE, where we hold the most expertise.

Java Magazine: What are your main concerns about the JCP?

Tene: We've been members of the JCP for about nine years. We joined the Executive Committee in 2011 in an effort to become more active in the process. Like the rest of the community, we had concerns about the shifts in the platform that might occur with Oracle's acquisition of Sun, so rather than sit on the sidelines and watch it happen, we decided to actively participate. Our concern is to make sure that the JCP is not dominated by a single company and that the JCP is not just a rubber stamp for Oracle—something that I believe Oracle does not want to have happen either.

ON INNOVATION

The JCP is certainly a place to have innovation happen.... But we should focus that innovation on the parts that are under our control and within the purview of what the JCP is intended to do.

That requires participation at all levels from JCP members—from executive participation shaping process, rules, and governance; from expert groups in standardization efforts; all the way to people developing code and contributing in different ways. We don't want to see Java become a single-company platform, which means the community has to contribute to the leadership.

Java Magazine: How should the JCP walk the fine line between encouraging innovation and creating stable standards?

Tene: The two are not necessarily contradictory, but sometimes they are. Personally, I think that staying within the scope that the JCP, as a standards and community body, is able to credibly work and be productive in is very important. You may have seen the recent discussions about the Social Media API, JSR 357, which was about the need for a standard Social Media API for Java that the Executive Committee rejected in an 8-to-5 vote. We took a position against it. I think the JCP should innovate in areas that are under its control and scope, and when it can assemble credible expert groups that lead the industry in the subject matter involved. But I also think the JCP should not attempt to innovate outside of those boundaries. Having the JCP try to standardize things that extend far beyond the Java platform and that are in a state of flux is, in my opinion, fundamen-

Tene and members of Azul's engineering team discuss a current project. All team members receive a white lab coat after four years at Azul.

Tene and an Azul engineer brainstorm while working with magnetic building tools.

tally wrong, and will just result in a standard nobody uses.

The JCP is there to move the Java platform forward, but in areas where industry efforts already exist to stabilize, model, and standardize behavior, the JCP's role is to follow and not to lead. Imagine the JCP trying to standardize XML while "in flux" and before the W3C decided what the standards should look like. The JCP is not the right place to standardize those, but it's certainly the right place to incorporate external standards with APIs for Java.

Java Magazine: How can we increase community participation in the JCP?

Tene: There are many ideas flying around. I think the JCP is certainly a place to have innovation happen—nobody else will innovate in the Java platform for us. But we should focus that innovation on the parts that are under our control and within the purview of what the JCP is intended to do. For example, if we want a future version of Java SE to have certain features with new syntax or new capabilities or APIs, or define new ways for containers to work in the Java EE space, the JCP is the only place to standardize that. We can prototype it and work within Java.net and within open source communities and play with implementations, but as far as defining the Java platform and its features, the JCP is it.

So tying into community development and innovation efforts and helping them naturally flow through the JCP's process is, in my opinion, our best shot at maintaining and increasing community participation while doing the job the JCP is meant to do.

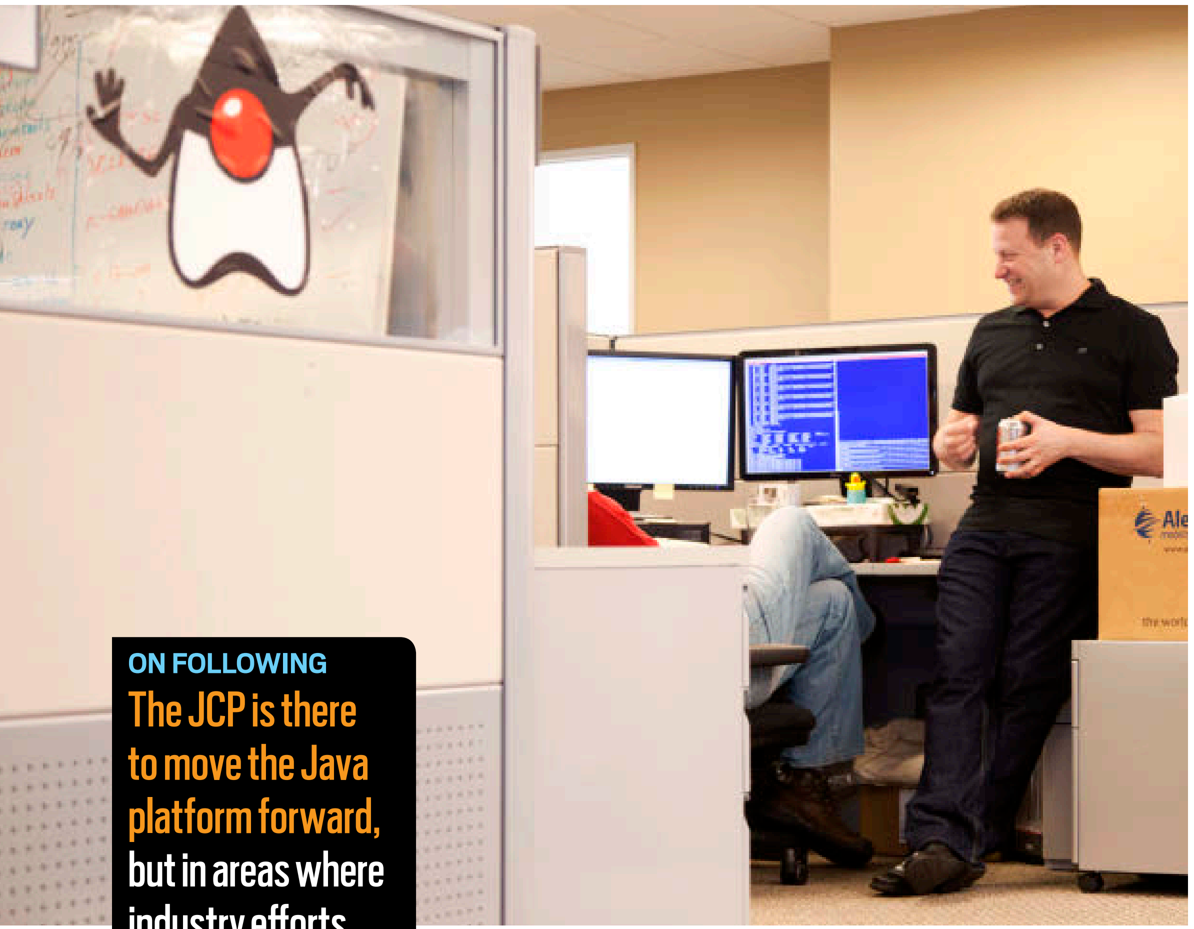
ON COMMUNITY

Tying into community development and innovation efforts and helping them naturally flow through the JCP's process is, in my opinion, our best shot at maintaining and increasing community participation while doing the job the JCP is meant to do.

Java Magazine: What are some key issues that the JCP needs to address?

Tene: There are clarification issues related to licensing, IP accessibility, and transparency that have been a concern for many people. The JCP has taken some good steps in the Executive Committee, and the JCP.next effort has begun addressing transparency and other process rules specifically with JSR 348. This leads to greater transparency in working groups. It's a good first step, but there are future second and third steps required in the JCP .next process. Some of these are structural and procedural, while some are more fundamental and about clarifying issues around accessibility of the actual implementations of the TCKs [technology compatibility kits], what the rules are, and what spec leads will do in order to provide access. This is dangerous territory because there are a lot of very strong opinions.

On the one hand, we have people who want to force the whole thing to be open sourced. I, for one, don't believe that's a viable path, and I think that an environment that allows solutions to take any form is the right way for a standard to work. Whether an implementation is proprietary, for profit, and highly controlled, or open source and free, we should be able to implement and adhere to the same standards.



Tene chats with a member of Azul's engineering team.

ON FOLLOWING
The JCP is there to move the Java platform forward, but in areas where industry efforts already exist to stabilize, model, and standardize behavior, the JCP's role is to follow and not to lead.

So I don't think that the JCP should be dictating specific license terms or saying specifically that certain licenses are allowed or not. But I do think that some guidelines and some boundaries are needed. For example, reliable and lasting access to TCKs under a known, predictable set of terms is a fundamental need for companies, projects, and individuals to invest in implementing and following a standard under the JCP. We should clearly and strongly define the minimal requirements that JSRs should meet for providing such access. We've seen over the last few years a lot of stagnation.

Java Magazine: Has Oracle delivered on the promise of increased transparency and openness in the JCP?

Tene: I would say that Oracle is in the process of *delivering*. I wouldn't say it has *delivered*. We see a lot of promise and very good intentions, but it's still too early to say that we see actual results. As part of the community, we look at this as something we continuously have to watch. Part of our role in the Executive Committee is to be that external, non-Oracle watchful eye that points out when we're heading in the wrong direction.

Let's be specific. We certainly have seen the JCP deliver on JSR 348 and improved rules and improved transparency so that community efforts will match what the community wants. But the process is only about half done. We still have more work to do in relation to things like the JSPA [Java Specification Participation Agreement] document that will be addressed in a future JSR.

The specific areas that we are concerned with are in-flux situations where the new rules that we've put in place actually contradict the previous way we've been working. So things that were fine a year ago under the previous process are now in contradiction with the new rules. And some of that has not yet been resolved. New rules that control what we can or can't do can stand in the way of doing work that we were previously able to do. For example, including confidential information in the materials discussed and worked on by JSR expert groups stands, in my opinion, in contradiction with the transparency rules and requirements of JCP 2.8 and JSR 348, but the JSPA in some cases allows and sometimes even requires expert group members to maintain levels of confidentiality around work that is material to a JSR. This sort of contradiction can potentially bring JSR work to a halt if we do not resolve it through rule changes and the JCP.next efforts.

Java Magazine: Do you think there's a perception in the IT community that Java is aging or that some other language or platform will replace it soon?

Tene: Yes, I think that there is a perception that Java is aging. There's a lot of talk about it slowing in innovation. The interesting thing is that one usually hears this sort of thing in the context of some new technology that is actually overtaking and replacing an older platform. But I think that this is not the case with the Java platform. The Java platform has been around for 17 years and has been enormously successful. Two or three years after Java first emerged, it was already clear that it was displacing other development and deployment platforms. I don't currently see some other emerging platform that is threatening Java like Java threatened other platforms. There are many interesting new developments in dynamic and functional languages, rapid development techniques, and other innovations that are not necessarily in the Java language, but the vast majority of those tend to target and run on the Java platform rather than threatening it. </article>

Janice J. Heiss is the Java acquisitions editor at Oracle and a technology editor at *Java Magazine*.

LEARN MORE

- Java Community Process

Data Quality Tools for Java

Telephone Verification
Name Parsing & Genderizing
Geocoding
Address Verification & Standardization
Email Address Verification
Web Services & APIs
Duplicate Elimination
MELISSA DATA

BEFORE

john smith iii phd
melissa data corp.
22382 Empresa 92688
7145895200
john@800miAL.con

AFTER

Melissa Data Corp.
John Smith III PhD
22382 Avenida Empresa Ste 100
Rancho Santa Margarita, CA 92688-2112
949-589-5200
John@melissadata.com
Delivery Indicator: Business

*Highlights indicate added and/or corrected data.

Realtime NCOALink Change-of-Address Web Service available

Now, finding the right data verification tools doesn't have to be so puzzling. Melissa Data offers customizable APIs, Web services and enterprise applications to match your budget and business needs. For solutions to cleanse, validate and standardize your contact data, we're ready to help you find the perfect fit.

Multiplatform

- Global address verification for 240 countries
- Clean and validate data at point-of-entry or in batch
- Correct misspellings, missing directionals, and confirm deliverability
- Enhance addresses with County, Census, FIPS, etc.
- Append rooftop lat/long coordinates to street addresses
- Update records with USPS and Canadian change of address info

Request free trials at MelissaData.com/myjava or call 1-800-MELISSA

MELISSA DATA®
Your Partner in Data Quality

Building a Better Spreadsheet

Java powers analytic breakthroughs at **QuantCell Research**.

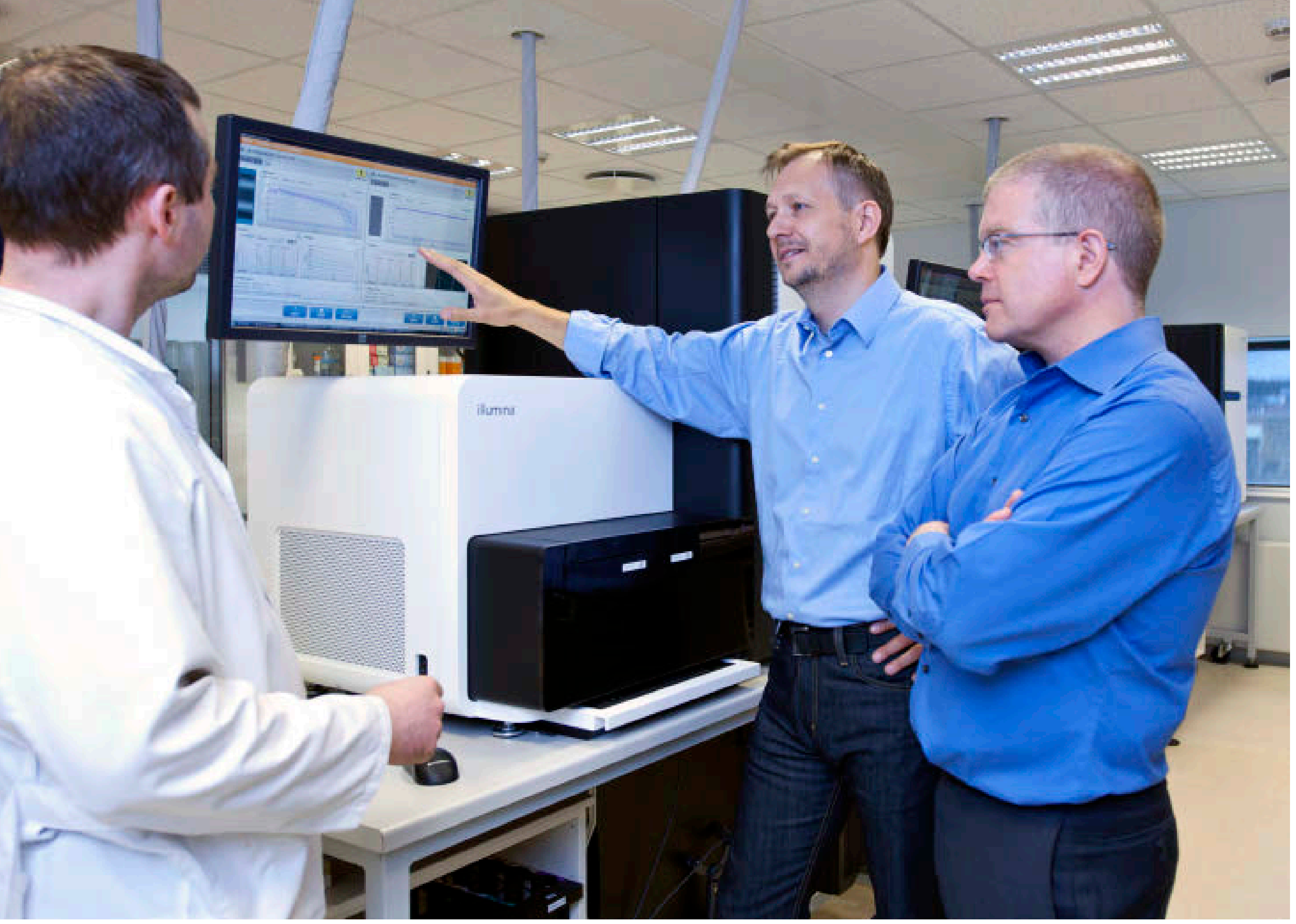
BY DAVID BAUM

(Left to right) Agust Egilsson, Kris Thorleifsson, and Bjorn Jonsson at QuantCell Research headquarters in Reykjavik, Iceland

If you want to change the world, you need to start with a mission that everybody can understand. Similarly, if you want to change the face of analytics, you need a simple paradigm that people can quickly comprehend. You don't invent a new programming language or craft a radical user interface; instead, you leverage the existing productivity tools that people know and love.

It was precisely this philosophy that motivated Agust Egilsson to develop QuantCell, a software environment for constructing visually rich models and big data analytics and applications. Rather than devising a new programming motif, Egilsson adopted one that animates businesses everywhere: the familiar spreadsheet interface. And instead of writing a new integrated development environment, he appropriated Java, the most versatile programming language of the modern age. The result is a powerful end-user environment that lets researchers, analysts, domain experts, and developers analyze any type of data in a fast, compelling way.

PHOTOGRAPHY BY THORSTEN HENN



Thorleifsson and Egilsson discuss QuantCell capabilities with a team member.

SNAPSHOT

QUANTCELL RESEARCH
quantcell.com

Headquarters:
Reykjavik, Iceland

Industry:
Technology

Java version used:
Java SE 7

"Observing quantitative analysts in the financial services industry, it was clear that people preferred to use spreadsheets for everything that they could," Egilsson explains. "So we set out to build an analytics and data visualization environment that preserves the benefits of the spreadsheet, such as short turnaround times, and addresses all of its shortcomings, such as the need to rely on the IT department for advanced functionality."

It's an area that Egilsson knows well. After receiving his doctoral degree in mathematics from the University of California at Berkeley, he delved into quantitative research and risk management for financial services institutions and biotechnology firms. He was soon joined in this endeavor by Kris Thorleifsson, a former Java product manager at Sun Microsystems with a background in cloud computing. After earning his MBA from Houston, Texas–based Rice University, Thorleifsson had served as CMO of the Dohop.com travel search engine and held senior management positions in the financial services industry before signing on with Egilsson to found QuantCell Research.

The pair had big ideas and an even bigger goal: to build a better risk-analysis system following the near collapse of the global financial industry. As the principal author and developer of the QuantCell environment, Egilsson had already been working with a prototype of the QuantCell system, both in investment banking and academia. Now he set his sights on one of the holy grails of computing: a programming environment that would allow nondevelopers to create powerful models and applications on par with those created by IT experts.

Victor Grazi, a vice president at Credit Suisse in New York, New York, has been testing an alpha version of QuantCell as a potential way to simplify corporate valuation exercises. Grazi has been a Java developer since 1995 and has worked in the financial industry since 2000.

"In the Credit Suisse Holt division, we work with fund managers to analyze their portfolios and make recommendations," Grazi explains. "Our valuations yield thousands of variables, including statistics about a company's financial statements, projections, and ROI."

Typically a programmer must sit down with an analyst to plug all these variables into a custom program, but Grazi says QuantCell allows people with very little programming experience to achieve the same results within an intuitive spreadsheet environment.

"You can include libraries in the QuantCell spreadsheet and link cells to generate a tremendous amount of statistics," Grazi explains. "All of a sudden an analyst becomes a programmer. We have our own visualization and charting routines that users can access just by double-clicking on a cell. It's very convenient."

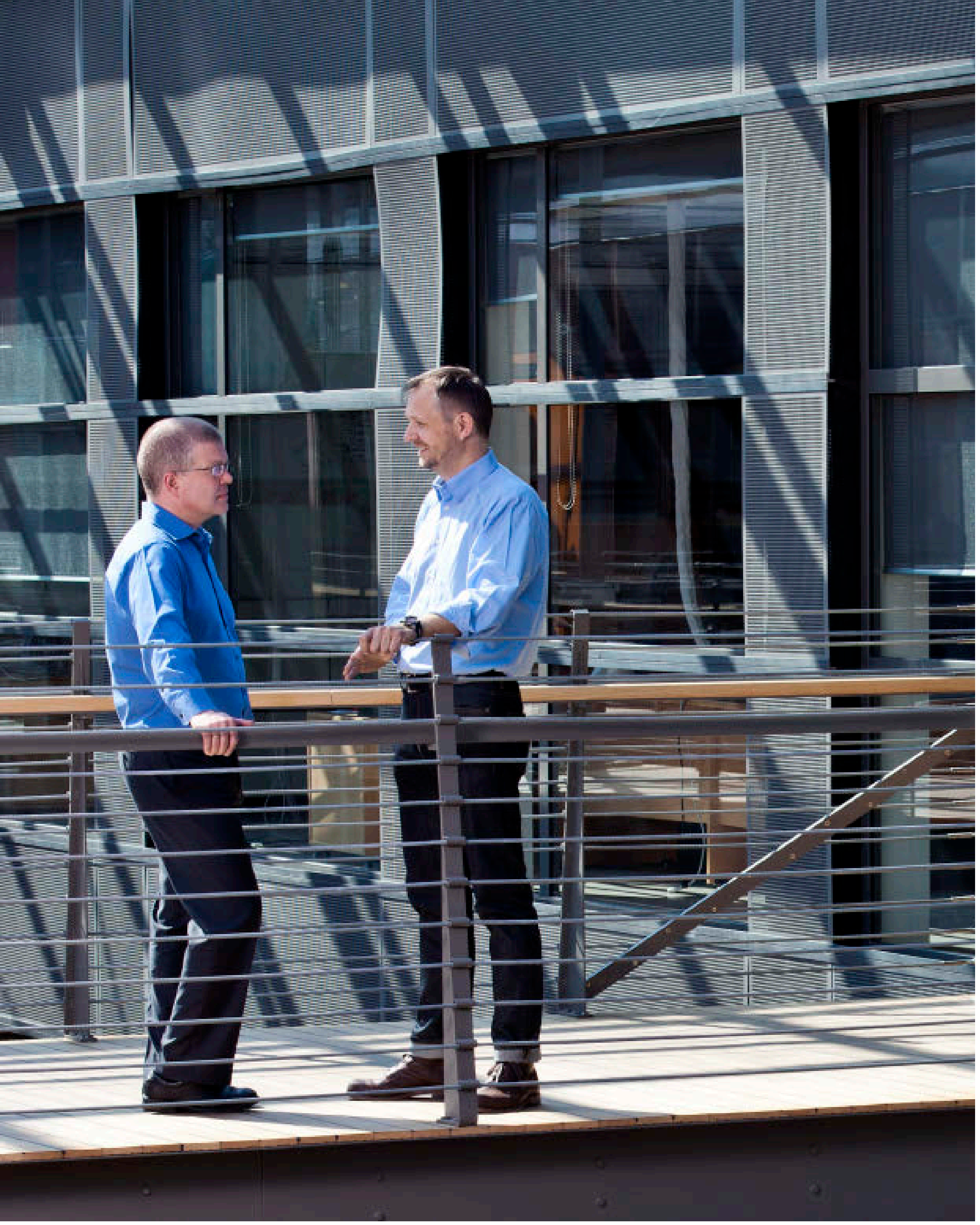
YEARS OF RESEARCH

"We are opening the door for nondevelopers to take advantage of many years of research and Java coding by some of the world's best researchers and institutions."
—Agust Egilsson, Cofounder, QuantCell Research

A FLEXIBLE FOUNDATION

By basing their solution on Java, QuantCell's founders gained instant support from an extremely large, well-informed, and generous developer community—aggregating a huge portion of the world's quantitative knowledge into their user-friendly environment. Thanks to the rich and ever-evolving Java ecosystem, hundreds of thousands of algorithms, processes, and methods are available for use with the QuantCell system.

"QuantCell can execute Java code snippets within spreadsheet cells, either entered directly by the user or created using end-user formula wiz-



Egilsson and Thorleifsson catch up on company business outside of QuantCell Research headquarters.

ards," says Thorleifsson. "The code is assembled, tested, and compiled one expression at a time, improving quality, shortening development time, and alleviating user-interface design chores."

QuantCell also enables direct access to some of the world's most important data providers, such as Thomson Reuters, Bloomberg, and Xignite—allowing people to search for data and make that data available for further analysis with one click.

The QuantCell environment creates Java classes or Java archive (JAR) files that can be deployed directly into any Java production system or are deployed automatically to cloud computing infrastructures such as Apache Hadoop. Unlike the lock-in typical with other systems, QuantCell models are independent of the environment and support open standards. They can also connect to their own private relational databases or semistructured datasources.

HOW IT WORKS

Novice users interact with QuantCell much like an online spreadsheet application, using UI wizards that generate Java, SQL, Python, or R-like expressions in conjunction with a rich knowledgebase of methods. The expressions in turn create Java objects in the spreadsheet cells such as mappers, reducers, tables, databases, images, multimedia, data cubes, compute clouds, and Java classes. Users visualize the objects via various UI controls such as charts and graphs, a table frame control, a database browser, a data cube viewer, a 3-D protein image viewer, a chemical structure visualization control, a spatial image control, and many other specialized components.

"We didn't initially understand what QuantCell was, but as soon as we figured out that the cell is actually the variable that contains the object, we began to see its potential," Grazi notes. "You can put any Java object into a spreadsheet cell and then refer to that object through another cell. To do something similar in another analytic environment is much more complex."

Experienced users can write their own functions to return objects to the cells, selecting components from a knowledgebase of externally and internally created Java libraries. For example, the company is creating a component browser to access pricing components for complex financial products and derivatives such as algorithms, payoffs, and simulation components. Advanced users can develop and deploy Java solutions as Web services as they create computationally intense projects requiring access to various datasources, algorithms, and computational clouds.

"Models and apps built in QuantCell are pure Java APIs that can be deployed into existing Java-based production systems or as Web services," explains Egilsson. "They run on the Java virtual engine and are therefore easily deployed into existing production systems as a Web service. Some of our early adopters are particularly excited about the ability to deploy custom MapReduce algorithms directly to Hadoop cloud installations from the spreadsheet interface."

ENRICHING THE BREW

QuantCell inherits its visual characteristic from JavaFX. Egilsson says he and Thorleifsson decided to use JavaFX because it lets people easily access existing Java APIs and because of its support for rich graphics, animations, and effects. "JavaFX is instrumental in bringing rich visualization methods to the spreadsheet," he says. "It's also universal, since it runs on the world's most widely used operating systems and supports all the major browsers."

In addition, JavaFX takes advantage of the computational power available locally on each user's machine, and each user can run complex computations locally in the Java

Agust Egilsson talks with author David Baum about QuantCell.

Virtual Machine (JVM) instead of relying on remote servers. For instance, users can view a 3-D image of a protein or an oil field, which would be less robust if the application were only server-based. "JavaFX delivers the performance we need and provides excellent UI controls right out of the box," Egilsson adds.

INDUSTRY APPLICATIONS

QuantCell Research is initially marketing its solution to finance professionals immersed in portfolio and risk analysis, because the solution lets those users take advantage of existing Java-based "quant" libraries—along with other Java-based libraries, such as the MapReduce framework for big data analytics. This versatile architecture lets them easily build advanced models and financial products faster and cheaper, and with less reliance on IT.

The company is also expanding into the life sciences industry, where alpha and beta users have found QuantCell ideal for analyzing genetic sequences, proteins, and markers.

"QuantCell's spreadsheet-based environment is a convenient way to engage nonexperts in complex analytical research," says Dr. Styrmir Sigurjonsson, a senior statistician at Natera, a life science company based in Redwood City, California, involved in prenatal genetic testing. "A statistician can develop algorithms based on lab data and then pass the QuantCell spreadsheet back to the lab researcher to continue the analysis. You can get to a whole new level of complexity without having to bog down the researchers with that complexity."

Natera currently uses MATLAB, a popular numerical computing environment and programming language, to develop its prenatal testing products. Sigurjonsson likes MATLAB's flexibility, saying researchers can "stop in the middle of a program, run other programs, and manipulate the data any way they like." He has been testing an alpha version of QuantCell to see if it can fulfill this same role in a Java environment. So far, he likes what he sees.

"Productizing something created in MATLAB is not a trivial process, but QuantCell will make the deployment more convenient," Sigurjonsson explains. "For example, in one cell you can get the data; in the next several cells you can analyze it; and then you can pull in off-the-shelf statistical packages, libraries, and visualization tools. You can develop an algorithm and deploy a JAR file right out of the system, so you have a deployed version that is ready for testing. I think it can save significant amounts of time in productization—perhaps as much as 60 or 70 percent."

QuantCell also supports long-running concurrent operations, allowing researchers to continue to work in a spreadsheet while the system crunches data from either a local database or a remote cloud.

BRIGHT FUTURE
"It is early on, but we are excited about QuantCell's potential."
—Victor Grazi, Vice President, Credit Suisse

COMMUNITY SUPPORT

As QuantCell gains momentum in anticipation of a production release later in 2012, the company is banking on continued support from the Java community to deliver a large and robust ecosystem of computational solutions.

"We decided to use Java in large part due to the dynamic nature of Java compilation, lazy class loading, concurrency in Java, and optimization in the VM such as just-in-time compilation," says Thorleifsson. "The many available libraries and the Java Community Process will keep this community thriving for a long time."

It's already a thriving ecosystem, fed largely by Java experts at universities and open source communities, which continue to contribute advanced analytic libraries. Popular examples include artificial intelligence and machine learning (Apache Mahout, Java-ML, and Weka), biotechnology (BioJava), and financial mathematics (jQuantLib). Unfortunately, most of these libraries are not immediately usable by biotech researchers, financial analysts, and other domain specialists, who must depend on skilled programmers to build custom applications for each analytical project.

QuantCell changes all that. All Java APIs, libraries, and tools work with QuantCell right out of the box. End users can call upon these libraries just by adding them to the classpath, expanding QuantCell's existing catalog of libraries, algorithms, and methods.

Applications and APIs created in NetBeans or Eclipse can be moved into the QuantCell environment for further work or testing, and those assembled in QuantCell can be moved into NetBeans for further development.

In short, QuantCell finds itself at the intersection of usability and functionality as it helps people of all skill levels take advantage of these rich analytic libraries on the one hand, and advanced data visualization tools on the other. </article>

David Baum is a freelance business, technology, and lifestyle writer in Santa Barbara, California.

For online travel leader **priceline.com**, Java provides the maximum connectivity, flexibility, performance, and portability.

BY PHILIP J. GILL

Direct Connection

Making online travel reservations seems straightforward enough: The prospective traveler enters a desired itinerary and is typically presented with a screenful of options, including choices, prices, photos, and more—all in a matter of seconds. From these options, the traveler makes a decision. Simple enough, right?

PHOTOGRAPHY BY CATHERINE GIBBONS

SNAPSHOT

PRICELINE.COM
priceline.com

Headquarters:
Norwalk, Connecticut

Industry:
Online travel services

Revenue:
US$4.36 billion in 2011

Employees:
350 in the priceline.com business unit, 3,400 worldwide

Java version used:
JDK 6 and JDK 7

For online travel sites, however, that up-front simplicity belies the complexity behind the screen. At priceline.com, for example, a simple query sets off a complex series of hundreds of interactions and connections to multiple hotel, airline, and other reservation systems around the world, explains Michael Diliberto, CIO for the North America division of the Norwalk, Connecticut–based firm.

"We work with thousands of suppliers and affiliates over multiple distribution platforms," says Diliberto. "We need to connect our system to the appropriate supply inventory, so that means direct connections to rental car companies such as Hertz, Avis, Alamo—and to large global distribution systems [GDSs] such as Travelport or even a smaller new GDS such as Farelogix, to get access to the freshest inventory for hotels, rental cars, and airline seats."

"A search for hotel deals in New York City can generate up to 500 simultaneous requests to suppliers' systems," Diliberto adds. Multiply that by thousands of customers querying the system simultaneously, and it's easy to see that the core of priceline.com's business is its ability to manage the complex matrix of connections between priceline.com's customers and suppliers. And the secret to making those connections work so seamlessly and to making it all look so easy in near real time is Java. "For us, Java has become a way of life," says Diliberto. "Java provides us the ability to connect with other travel resources around the world more effectively and efficiently."

BEHIND THE SCREENS

Priceline.com, the online travel business that made "Name Your Own Price" famous, is one of the world's leading online travel services, offering everything from hotel rooms to airline tickets, rental cars, vacation packages, and even cruises. It was founded in 1997 and a year later launched its travel services with

Priceline CIO Michael Diliberto (left) and Vice President of Engineering Amit Poddar discuss project status in front of the company's analytics dashboards.

the help of celebrity pitchman William Shatner.

In the late 1990s, there were a number of technologies available for building dynamic, interactive Websites. These included Java as well as Perl, PHP, and Microsoft's COM and Active Server Pages (ASP). Although Java was less mature—it had only been introduced in 1995—than the others, Diliberto saw great potential in the language and the development platform, not the least of which was its promise of portability and vendor independence. In the end, the company chose a three-tier design: Java at the back end, Oracle for database management, and COM and ASP at the front end.

NOT SO SIMPLE

A single user query on priceline.com can generate up to 500 simultaneous requests to suppliers.

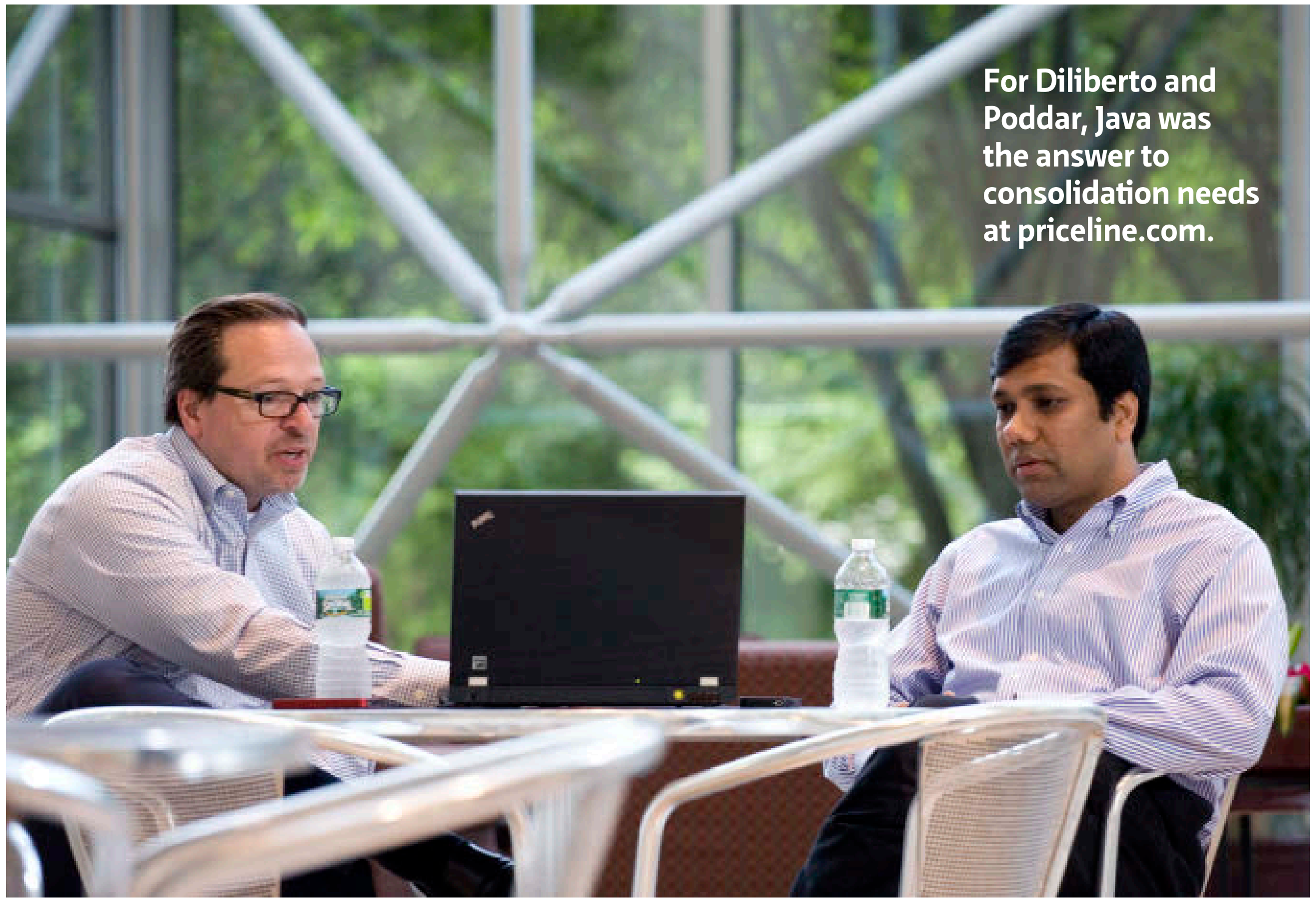
According to Diliberto, Java was ideal for the back-end inventory search engines because of its exceptional connectivity and multithreading capabilities.

"Right from the very beginning, our back-end communications to our suppliers has been designed and developed on a Java platform," says Diliberto. "Java supports a variety of different communication protocols to interface with suppliers, so that we can go where the data is—in our case, connecting to GDSs, airlines, hotel partners, and chains—to get the freshest inventory at the best price."

The combination of Java and Microsoft technologies worked fine for the first few years, but priceline.com had a change of heart in 2000.

"We realized that having a different technology platform for front end and back end was not the way to go," explains Amit Poddar, vice president of engineering at priceline.com. "We needed to consolidate on one platform to avoid unnecessary overhead of data transformations and to utilize our talent pool more effectively."

The priceline.com team evaluated multiple enterprise platforms for performance, interoperability, and availability of tools and programmers. Java was a clear winner. Since then, all major enhancements to the priceline.com Website have been coded in Java.


For Diliberto and Poddar, Java was the answer to consolidation needs at priceline.com.

SEARCH SMARTER, NOT HARDER

Over the years, priceline.com grew from its travel bidding model into a full-service online travel company that includes published price and Name Your Own Price travel services. "Priceline.com is the only travel site on the internet to offer both options for travelers," notes Diliberto.

To present various travel options to their consumers in real time, priceline.com developed a proprietary "smart search" system using Java's multithreading framework. The system acts like an inventory switch, says Poddar. "On one side are our customers, including Priceline-owned Websites, affiliates, search engines such as Google, and more recently mobile phones and tablets such as iPad," he continues. "On the other side are hotels, airlines, car rental companies, and GDSs such as Travelport, Sabre, and Pegasus, as well as startups like Farelogix."

When a customer is searching for the best hotel deals on priceline.com, says Poddar,

Oracle and Java: A Good Marriage

Priceline.com has been committed to both Oracle and Java technologies since the Website was launched; indeed, today the site's core production transactional database is a 2 TB Oracle database, and it also maintains a 17 TB Oracle data warehouse. In total, the Norwalk, Connecticut–based online travel agency has 96 unique instances of Oracle Database in its IT shop, a configuration that provides both high performance and high availability.

That's why Michael Diliberto, the company's CIO, North America, sees Oracle's acquisition of Sun and its Java technologies as a good deal for both Java users and the future of the platform.

"From the beginning, Oracle was committed to the internet as a business platform," says Diliberto. "[Oracle CEO] Larry Ellison saw the power of the internet and knew it was going to be a major piece of the economy going forward, and he wanted to prepare for it. And from a database standpoint, there was always a solid commitment to creating the biggest, best-performing, and most scalable database there could be."

"We've always been big fans of Java as a technology but had always been a little bit concerned about its survival," Diliberto continues. "With Oracle having a solid track record of turning products into profit, we've been encouraged that Oracle's leadership sees a future in Java and is willing to invest in it and drive it forward, to keep it alive and to keep it growing. The company has proven that over the last two years. We're very happy with the continued investment that Oracle has been making in the Java platform."

behind the scenes the priceline.com smart search concurrently fetches data from various suppliers in real time, sorts and filters the combined data, and presents the best options to the customer.

"It's very important that the inventory we are presenting is fresh," Poddar adds, "because there's a lot of competition in the online travel space, and if a customer finds that our inventories are stale, they'll go to another travel site and book there."

Not all suppliers have the same interface, Poddar explains, but Java supports a vast array of data formats, transformation tools, and communication protocols.

"With Java we can easily receive data from suppliers either in SOAP wrapped XML using HTTPS or in structured data records over JMS," he says. "We can then transform that data using JAXB or other Java APIs into Java beans. Then we compare those beans to hundreds of other responses and create a single response formatted in one of several presentation mechanisms. We could send the response as a JSON object consumed by Priceline's app on a customer's iPad, or ship it in XML format to one of Priceline's affiliates."

For this reason, adds Poddar, "Java turns out to be the perfect technology for a flexible inventory switch."

GREAT MIGRATION
Priceline.com has migrated its Java applications across three different hardware architectures and three different operating systems.

VENDOR INDEPENDENCE

Vendor independence was a key criterion when selecting technology as well, and that's where competing technologies fell short. "Instead of building a specific product, Java defines a specification and lets multiple vendors in the industry provide an implementation. This gives choice to customers," says Poddar.

From the start, priceline.com's back-end database has been an Oracle database. But back in 2000, Oracle didn't have what priceline.com considered a viable Java driver for that database. "We started using drivers from one company," says Poddar. "The performance was OK, but we didn't really like it, so we moved to a second company. Eventually Oracle came out with their own drivers and we moved to them."

"We were able to successfully replace one implementation of the driver with another without modifying a single line of code," Poddar says. "And that was possible because of Java's philosophy that you define a specification and let the vendors do the implementation. This puts the customer in full control and gives them the ability to find the right itinerary for them."

Over the years, as priceline.com's needs have changed, Java has also enabled the company to migrate its IT infrastructure across three different hardware architectures and three different operating systems with little effort, Poddar adds. The company launched in 1998 running on a Digital Equipment Corporation AlphaServer with Windows NT, then migrated to SPARC-based servers running Solaris (now Oracle Solaris), and today operates on an array of HP Intel–based blade servers running 64-bit Red Hat Linux.

"Java has enabled us to move from one hardware platform to another quite easily," says Poddar. "We didn't have to involve any of the Java programming teams for the migration. It was completely done by the support staff."

Another important benefit is that Java is open source. "Priceline.com is committed to open source technology wherever and whenever possible as a way to maintain vendor independence," explains Diliberto.

Besides Java, the company's IT infrastructure uses the Apache Tomcat Web server and the Linux operating system. "Going with open source technologies such as Java as much as possible puts us in a good position," says Diliberto. "We feel that it gives us an opportunity to select implementations that are best of breed and the most effective solutions." </article>

Philip J. Gill is a San Diego, California–based freelance writer and editor.

MICHAEL KÖLLING

BIO

Part 1

Learning and Teaching Object Orientation with BlueJ

A systematic and experimental approach to learning Java

In the previous issues of *Java Magazine*, we went step by step through a project of building a simple game in Java using the Greenfoot environment. Greenfoot is a great tool to engage young learners by getting them highly motivated very quickly. The ease with which we can put animated graphics onscreen is a fantastic help in drawing kids to programming.

This time, however, we want to look at approaching the learning and teaching of object-oriented programming more systematically. To support this, we will use another tool: BlueJ.

The strength of Greenfoot is a quick, playful entry to programming with immediate visual results. The drawback, however, is that there is a bit of magic going on: some of the code is provided by the environment (the runtime framework), and the type of program we can create is restricted to two-dimensional graphical applications.

We now grow up a bit and shift to BlueJ. BlueJ is a generic development environment—no code is magically provided, and we can develop any kind of application we like. Thus, in its purpose, it is closer to well-known professional IDEs, such as NetBeans and Eclipse. However, it is still an IDE focused on learning object orientation, and for a variety of reasons, it is much better suited to beginners than large professional environments.

In this article, I will provide an overview of BlueJ, which is most useful for those of you who can already program and are looking for a tool to teach programming to beginners, either at the university level, toward the end of high school (for example, in an AP computer science course), or in an after-school programming club. I will give you an idea what BlueJ is and what it provides.

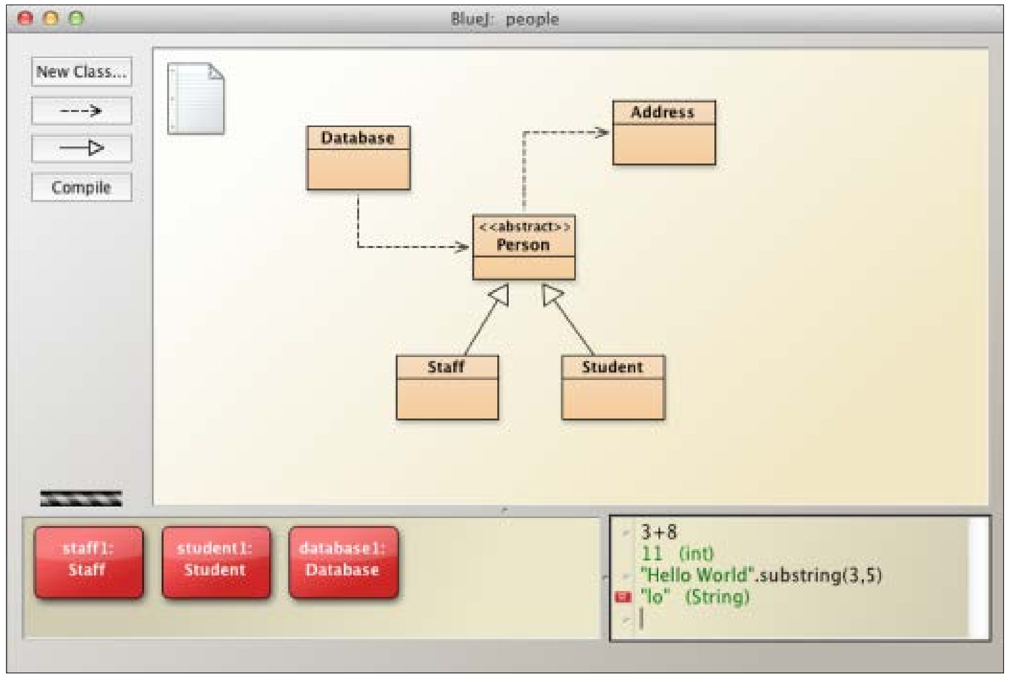
The best way to read this article is to download BlueJ and install it. Then download the "people" project, and play along as we go.

In future issues of *Java Magazine*, I will go through a small project that can be used to learn object-oriented programming with BlueJ and Java.

Simplicity

The first question many people ask when I talk about teaching object orientation and Java to beginners is, "Why not just use NetBeans or Eclipse or (insert your own favorite environment here)?" The answer is that the requirements for an environment for beginners are significantly different than the requirements for an environment for professional programmers.

Many software tools that are useful for professionals represent only clutter, confusion, and unnecessary hurdles for begin-



```
3+8
11   (int)
"Hello World".substring(3,5)
"lo"   (String)
```

Figure 1

PHOTOGRAPH BY JOHN BLYTHE

ners. With big, professional environments, students spend a large amount of their brain power thinking about the *environment*, rather than thinking about *programming concepts*.

The first difference in BlueJ is its simplicity (see **Figure 1**). Students feel comfortable using the environment very quickly, and we never teach about the environment; we teach about concepts. You will see what I mean when you start BlueJ yourself. After only a few hours, students are entirely comfortable using the environment.

The Toolset

The second difference is the toolset that is provided. Not only does BlueJ provide *fewer* tools than professional IDEs, it provides *different* tools. The tools provided by BlueJ encourage and support visualization, interaction, and experimentation. They were designed by experts in programming education with educational theories in mind, and they allow a different style of learning than standard IDEs offer.

The tools include mechanisms for active, direct experimentation with objects. This is one of the most fundamental functionalities missing (from an educational viewpoint) in most environments. Let's have a look at the most important of them.

The Class Diagram

The first visual tool is apparent immediately when you open your first project: the class diagram (see **Figure 1**). It is the main view of the project and shows the classes in the project and their relationships. The diagram supports and reinforces thinking about class structures from the very beginning. In early teaching, students typically do not start with an empty screen. Instead, they are given projects (often partially implemented) to experiment with and extend.

Interactive Object Instantiation

Once a class has been written and compiled, we can right-click the class in the diagram to see a context menu. This menu offers any public constructor to allow us to manually create an object of this class (see **Figure 2**). If the constructor has parameters, a dialog box pops up to let us enter the necessary values.

Once an object has been created, it appears (in red) on the object bench, toward the bottom of the main window (see **Figure 1**). When teaching, it is important to create multiple instances of the same class—this makes it very easy for students to see the difference and the relationship between classes and objects: From a class, we can create an object. In fact, we can create many objects.

SUPER EASY
BlueJ provides probably the most convenient method of any environment for creating JUnit tests.

When using traditional environments, understanding the difference between classes and objects is one of the concepts known to be difficult for students. This is not surprising if all they ever look at is lines of code. The visualizations and interactions in BlueJ entirely remove this problem; students just "get it."

Object Interaction

Once an object has been created on the object bench, we can interact with this object by invoking its public methods (see **Figure 3**). Again, if a method has parameters, a dialog box allows us to enter their values. Return values are displayed for methods with results.

This easy manual interaction allows a very quick and thorough testing of code, which can be used to investigate a project to find out what it does or to test code that was just written to see whether it behaves as expected.

Because testing is possible without the need to write test drivers, the hurdle to testing is much lower than in traditional environments. Therefore, students typically test earlier and more often. Also, because results can be seen during a test sequence, further tests can be adjusted depending on the previous result. This is not the case if tests were executed from prewritten scripts.

Apart from better testing, the interaction reinforces the following most-fundamental principles of object-oriented programming:

- A program consists of a set of interacting objects.
- We communicate with objects by invoking their methods.
- Methods are provided by the objects.
- Methods may have parameters and return values.
- Parameters have types.

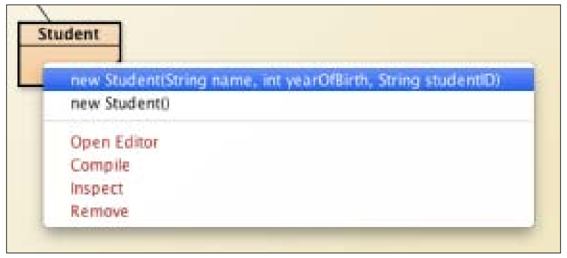


Figure 2

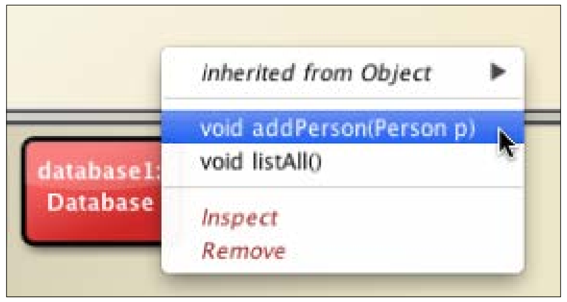
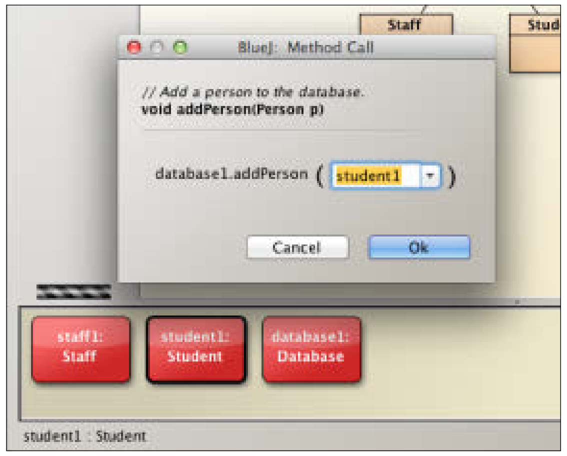


Figure 3



```
// Add a person to the database.
void addPerson(Person p)

database1.addPerson ( student1 )
```

Figure 4

By interacting with objects, students are conditioned to think in terms of objects from Day One. This is a fundamental reversal of traditional approaches that use older textbooks and standard IDEs.

Composition

Parameters to methods can be not only predefined or primitive types; they can also be objects. **Figure 4** shows an example where a Person object is expected. Any object present on the object bench can be passed as a parameter (as long as the type matches) by simply clicking it when the parameter dialog box pops up.

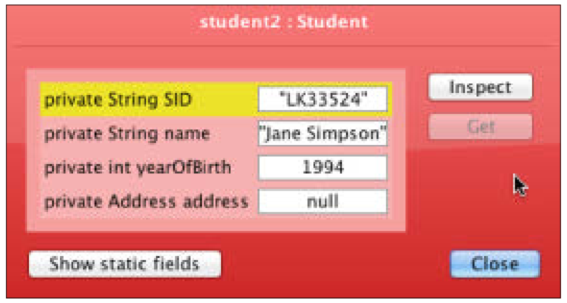
State

BlueJ offers an inspector capability that allows us to open an object and look inside it (see **Figure 5**). It is highly educational to look into two or three objects of the same class and observe their fields. We can see that they all have the same fields, but their values differ. We can even make calls to an object's methods while the inspector is open to observe the values changing. This reinforces another important concept: state.

Yes, Really: No main Method

If you played along until now, and you're used to writing Java programs, you will have noticed one thing: no main method. Yes, that's right: there is no main method!

The main method is an extremely ugly hack for connecting a Java program to the operating system. It has nothing whatsoever to do with object orientation and has confused students ever since it was first shown.



private String SID	"LK33524"
private String name	"Jane Simpson"
private int yearOfBirth	1994
private Address address	null

Figure 5

In BlueJ, we start with the clean principles and add the idiosyncrasies of the language later. You can, of course, have a main method and use it if you want to. It will appear (together with all public static methods) in the class' menu, where it can be invoked. However, it is nothing special and can be introduced if and when students are ready to really understand the meaning of public, static, void, and String[].

The Code Pad

Another very useful educational tool is the *Code Pad*. The Code Pad is visible at the bottom right in **Figure 1**. It allows us to type individual expressions or statements, which are then immediately executed. For expressions, the result is immediately displayed in the Code Pad. If the result is an object, it can be dragged across to the object bench for further investigation.

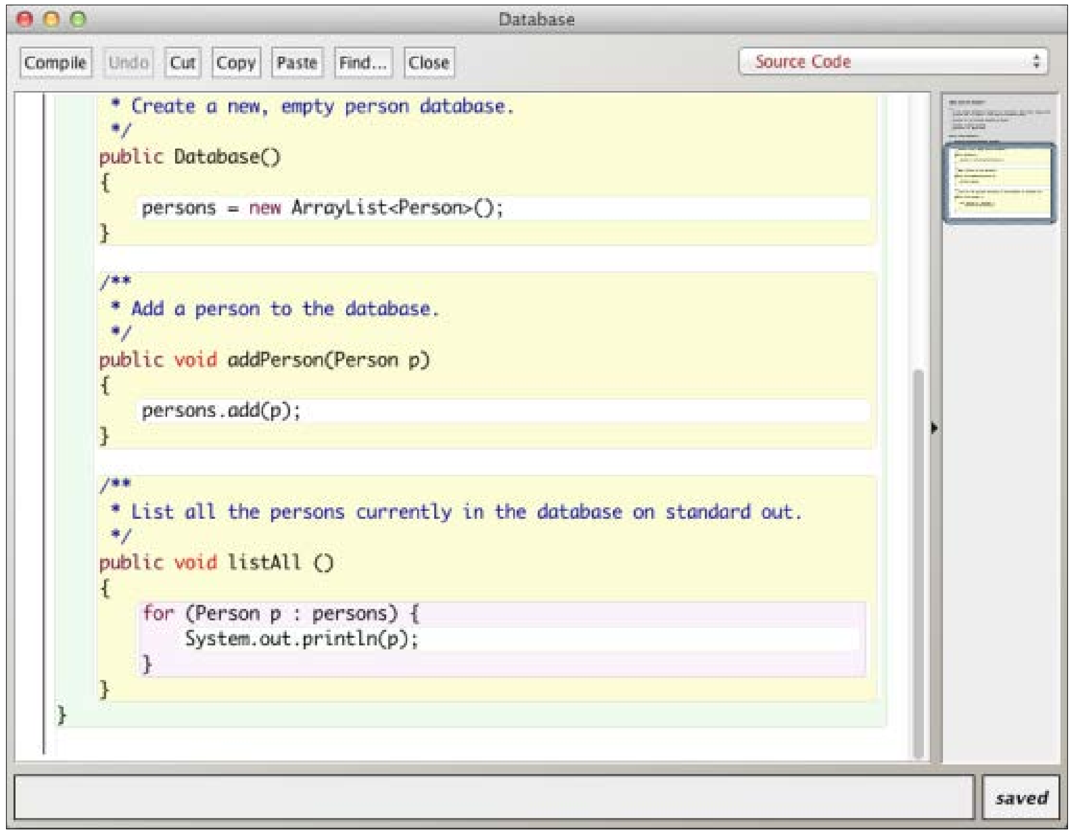
The Code Pad is very useful for students to quickly and easily try out arbitrary code snippets to see what they do.

The Editor

BlueJ's text editor highlights scopes, such as methods, if statements, and loops with different background coloring (see **Figure 6**). This is another educationally valuable visualization. The coloring supports the understanding of scoping, and errors in scope become much more easily apparent.

Unit Testing

Unit testing (using JUnit) is closely integrated in BlueJ. In fact, BlueJ provides



```
     * Create a new, empty person database.
     */
    public Database()
    {
        persons = new ArrayList<Person>();
    }

    /**
     * Add a person to the database.
     */
    public void addPerson(Person p)
    {
        persons.add(p);
    }

    /**
     * List all the persons currently in the database on standard out.
     */
    public void listAll ()
    {
        for (Person p : persons) {
            System.out.println(p);
        }
    }
}
```

Figure 6

probably the most convenient method of any environment for creating JUnit tests. In addition to the standard writing of test classes, interactive testing and regression testing can be combined. We can perform a sequence of interactive tests and record this interaction as a JUnit test case to be replayed later.

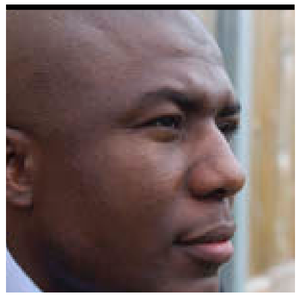
We will discuss this functionality in more detail in a later article in this magazine.

Conclusion

In this article, I presented a quick overview of the most important tools of the BlueJ environment. From the next article onward, we will look at a specific project, and discuss how these tools can be used to teach the fundamentals of Java and object orientation to beginners. Until then, get BlueJ installed on your system and start experimenting! </article>

LEARN MORE

- Java SE 7 API



MAX BONBHEL

BIO

Part 1

Introduction to Web Service Security from Server to Client

Secure your Web services with Metro, GlassFish, and the NetBeans IDE.

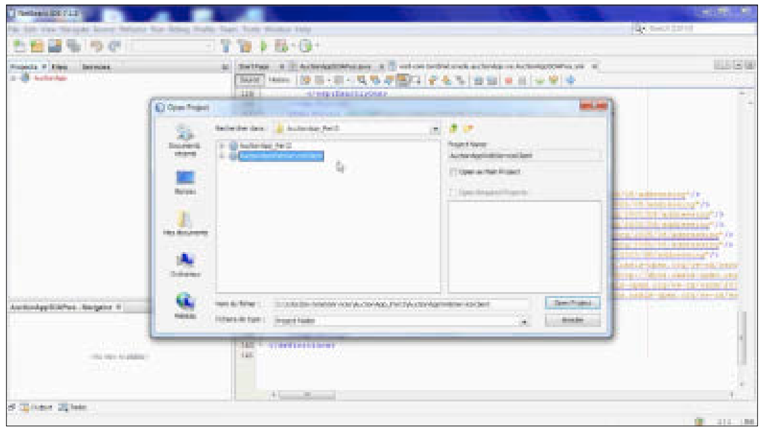
Web services are the best way to integrate new or extend existing functionality into an application, but this causes new problems for the security of data transiting between the client and the server. This article is the first in a three-part series that will focus on the different aspects of SOAP Web services, such as security, reliability, and transactions. The goal of this series is to highlight methods for increasing the security of applications within the context of increasingly complex systems.

In this article, I will demonstrate how to secure Web services efficiently both on the server and the client using Metro, GlassFish, and the NetBeans IDE.

Note: The complete source code for the application designed in this article can be downloaded here.

What Is Metro?

Metro (included in the NetBeans IDE) is a high-performance, extensible, easy-to-use Web service stack. It is a one-stop shop for all your Web service needs, from the simplest "hello world" Web service to reliable, secured, and transacted Web services that involve .NET services. The Metro platform provides, in one place, all we need to build production-quality Web services.



Author Max Bonbhel demonstrates how to secure your Web services.

PHOTOGRAPH BY ALLEN MCINNIS/GETTY IMAGES

What Is WSIT?

Web Services Interoperability Technology (WSIT) is a part of the Metro Web service stack along with the Java API for XML Web Services Reference Implementation (JAX-WS RI). It provides a complete architecture and tools for developing the next generation of Web service technologies. WSIT consists of Java APIs that enable advanced Web service features to facilitate interoperability with .NET.

Prerequisites

Download the following software, which was used to develop the application described in this article:

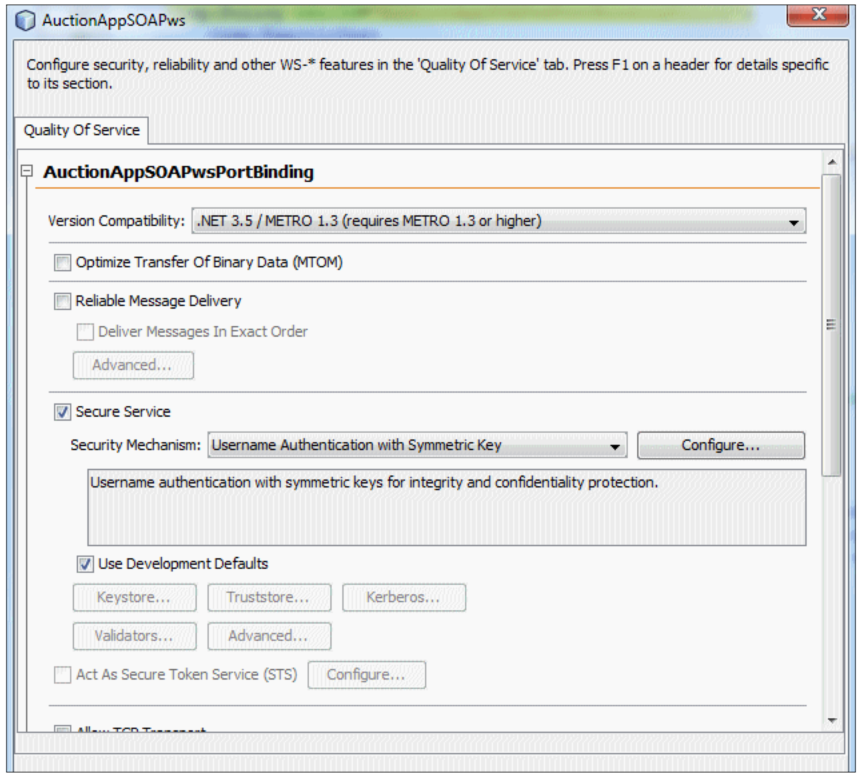
- NetBeans IDE 7.1.2 (available for download here)
- GlassFish 3.1.2 (available for download here)
- Metro 1.3 or higher (included in NetBeans)
- The AuctionApp project from Part 3 of "Introduction to RESTful Web Services" (available for download here)

Note: This article was tested with the latest version of the NetBeans IDE (version 7.1.2, as of this writing).

Overview of Adding Security Options to the Web Service

What we are going to do is secure the service and the client of the AuctionApp project we created in the previous series of articles ("Introduction to RESTful Web Services") by performing the following tasks:

- Secure the Web service:
  - Add a security mechanism called Username Authentication with Symmetric Key.
  - Import the certificates into the GlassFish application



AuctionAppSOAPws

Configure security, reliability and other WS-* features in the 'Quality Of Service' tab. Press F1 on a header for details specific to its section.

Quality Of Service

AuctionAppSOAPwsPortBinding

Version Compatibility: .NET 3.5 / METRO 1.3 (requires METRO 1.3 or higher)

☐ Optimize Transfer Of Binary Data (MTOM)

☐ Reliable Message Delivery

☐ Deliver Messages In Exact Order

Advanced...

☑ Secure Service

Security Mechanism: Username Authentication with Symmetric Key — Configure...

Username authentication with symmetric keys for integrity and confidentiality protection.

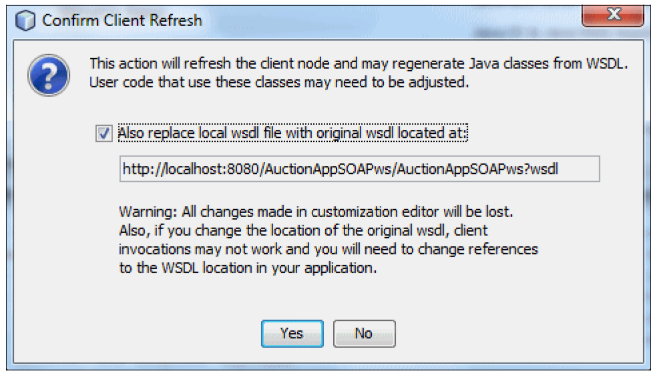
☑ Use Development Defaults

Keystore... Truststore... Kerberos...

Validators... Advanced...

☐ Act As Secure Token Service (STS) — Configure...

Figure 1



Confirm Client Refresh

This action will refresh the client node and may regenerate Java classes from WSDL.
User code that use these classes may need to be adjusted.

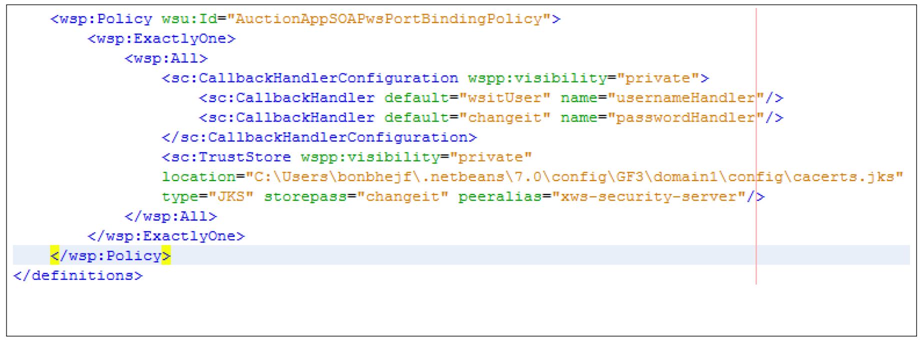
☑ Also replace local wsdl file with original wsdl located at:

http://localhost:8080/AuctionAppSOAPws/AuctionAppSOAPws?wsdl

Warning: All changes made in customization editor will be lost.
Also, if you change the location of the original wsdl, client
invocations may not work and you will need to change references
to the WSDL location in your application.

Yes No

Figure 2



```
    <wsp:Policy wsu:Id="AuctionAppSOAPwsPortBindingPolicy">
        <wsp:ExactlyOne>
            <wsp:All>
                <sc:CallbackHandlerConfiguration wspp:visibility="private">
                    <sc:CallbackHandler default="wsitUser" name="usernameHandler"/>
                    <sc:CallbackHandler default="changeit" name="passwordHandler"/>
                </sc:CallbackHandlerConfiguration>
                <sc:TrustStore wspp:visibility="private"
                location="C:\Users\bonbhejf\.netbeans\7.0\config\GF3\domain1\config\cacerts.jks"
                type="JKS" storepass="changeit" peeralias="xws-security-server"/>
            </wsp:All>
        </wsp:ExactlyOne>
    </wsp:Policy>
</definitions>
```

Figure 3

server and set up a default user for this application.

- Configure the client that references the secured Web service.
- Test the secure Web service:
  - Use the Username Authentication with Symmetric Key security mechanism.
  - Provide the certificates and use a default user credential to access the secured service.

The application we are going to secure is an online auction place (like eBay) that we created in the previous series of articles. Sellers post their items in listings, and buyers bid on the items. A seller can post one or many items, and a buyer can bid on one or many items.

Specifically, we are going to limit access to the JAX-WS Web service that extrapolates the amount of a bid.

Secure the Web Service

Let's secure the Web service in two minutes using the NetBeans IDE.

1. Add a security mechanism called Username Authentication with Symmetric Key in the AuctionApp application:
   a. Open the AuctionApp project in NetBeans IDE 7.1.2 or higher.
   b. Expand the **Web Service** node of the AuctionApp project and right-click the **AuctionAppSOAPws** node; then select **Edit Web Service Attributes**.
   c. Under the Quality Of Service tab, expand the **AuctionAppSOAPwsPortBinding** section (see **Figure 1**).
   d. Make sure the **Reliable Message Delivery** option is *deselected*.
   e. Select **Secure Service** and select **Username Authentication with Symmetric Key** from the Security Mechanism list.
2. Import certificates into the GlassFish application server and set up a default user, so we can use the server immediately to test our application:
   a. Select **Use Development Defaults** to import certificates and set up a default user, as shown in **Figure 1**.
   This option allows NetBeans to import certificates into the application server by creating an entry in the GlassFish keystore and truststore.
   b. Click **OK**.
   NetBeans generates the appropriate configuration for the Web service based on the options we selected.
3. Deploy the Web service by right-clicking the **AuctionApp** project node and choosing **Deploy**.

NetBeans creates a new WSIT configuration file that contains detailed information about the options and the runtime usage of the secured service. The file is generated in the Web Pages/WEB-INF node of the AuctionApp project.

Configure the Client

In this section, we are going to refresh and configure the Web service client that references the Web service that was secured in the previous section.

The Web service client will use the certificates that were imported and the default credentials to access the secured Web service.

1. Refresh the Web service client:
   a. Make sure the AuctionApp is up and running. If it is not, right-click the **AuctionApp** node and select **Deploy**.
   b. Open the **AuctionAppWebServiceClient** project in NetBeans IDE 7.1.2 or later. Expand the **Web Service References** node of the AuctionAppWebServiceClient project and right-click the **AuctionAppSOAPws** node; then select **Refresh**.
   c. From the Confirm Client Refresh wizard, select **Also replace local wsdl file with original wsdl located at:**, as shown in **Figure 2**.

d. Click **OK**.
At this point, the Web service client is regenerated from the corresponding Web Services Description Language (WSDL) file.

2. Open the AuctionAppSOAPws .xml file located in the Source Packages/META-INF node of the AuctionAppWebServiceClient project. The wsp: Policy tags in the AuctionAppSOAPws.xml file should look as shown in **Figure 3**.

3. Deploy the client application by right-clicking the **AuctionAppWebServiceClient** project node and choosing **Deploy**.

Test the Secured Web Service

Now it's time to test the service. We will invoke the secured Web service from the client application.

We are going to use a sample front-end JavaServer Faces (JSF) application to perform the following tasks:

- Try to invoke the secured Web service without providing the certificates and the default user.
- Use the certificates and the default user to invoke the secured Web service that extrapolates the amount of a bid.

1. Try to invoke the secured Web service to display the extrapolated amount of a bid:

a. Make sure the AuctionApp project we opened previously is up and running. If it is not, right-click the **AuctionApp** node and choose **Deploy**.

b. Expand the **Web Service References** node of the **AuctionAppWeb-ServiceClient** project and right-click the **AuctionAppSOAPws** node; then select **Edit Web Service Attributes**.

c. Under the Quality Of Service tab, expand the **Security** section.

d. Make sure the **Use development defaults** option is *deselected*, as shown in **Figure 4**.

e. Click **OK**.

f. Right-click the **AuctionAppWeb-ServiceClient** project and choose **Clean and Build**.

g. Right-click the **AuctionAppWeb-ServiceClient** project again and choose **Run**.

The list of all entries is displayed, as shown in **Figure 5**.

h. Click the **Show all Bid Items** link to display the list of bid entries, as shown in **Figure 6**.

i. Click the **View** link for the bidder named Vals to see the newly extrapolated amount of the Vals bid, as shown in **Figure 7**.

As you can see, the amount of the bid changed from 12.0 to 0.0. This means that the client failed to call the secured Web service.

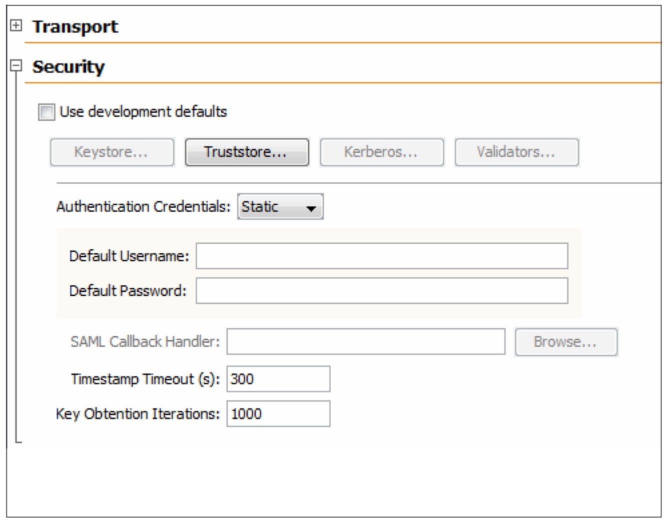

Transport
Security
☐ Use development defaults
Keystore... Truststore... Kerberos... Validators...
Authentication Credentials: Static
Default Username:
Default Password:
SAML Callback Handler: Browse...
Timestamp Timeout (s): 300
Key Obtention Iterations: 1000

Figure 4

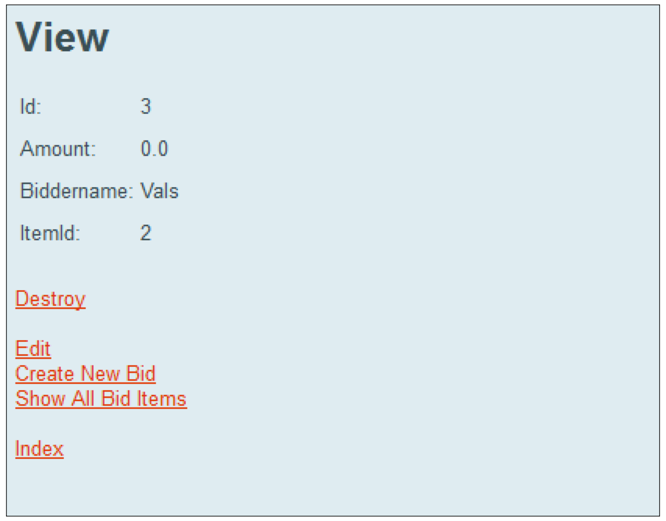

View
Id: 3
Amount: 0.0
Biddername: Vals
ItemId: 2
Destroy
Edit
Create New Bid
Show All Bid Items
Index

Figure 7

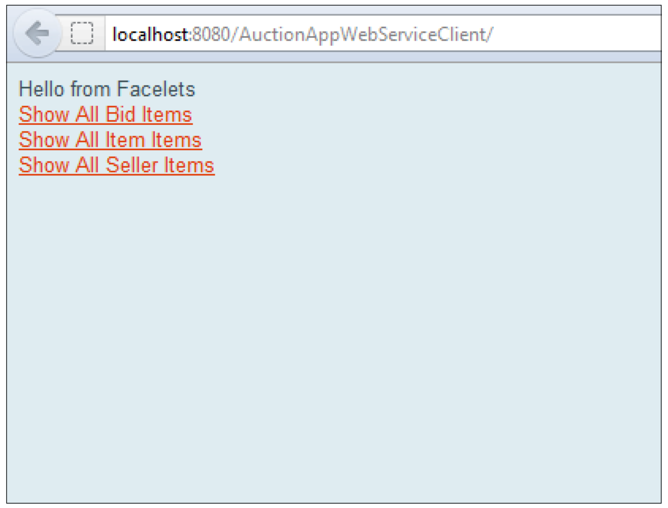

localhost:8080/AuctionAppWebServiceClient/
Hello from Facelets
Show All Bid Items
Show All Item Items
Show All Seller Items

Figure 5

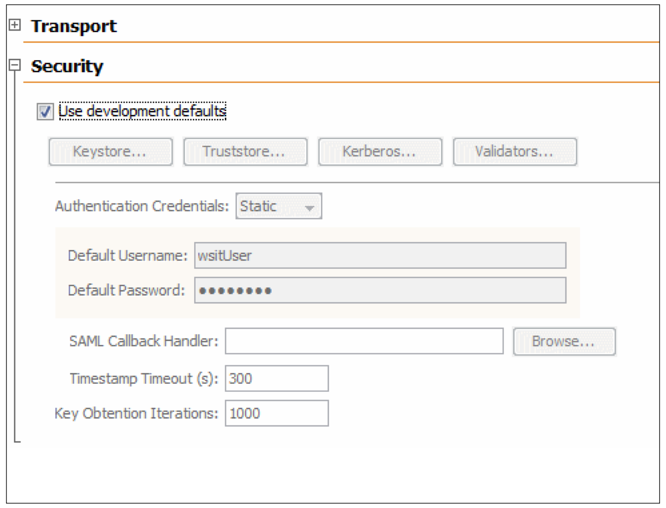
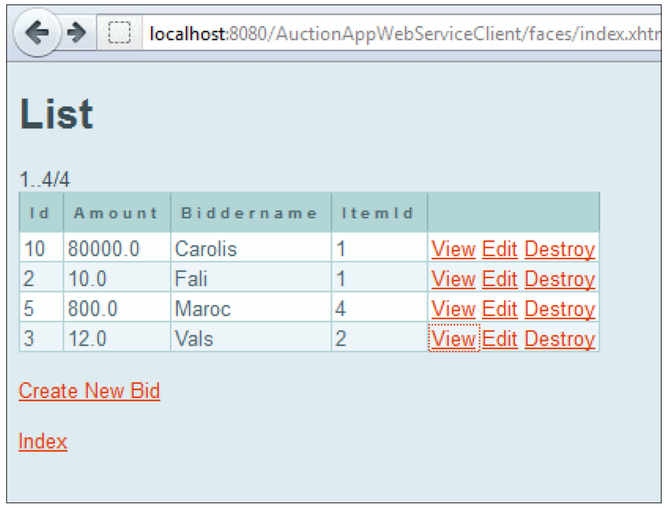

Transport
Security
☑ Use development defaults
Keystore... Truststore... Kerberos... Validators...
Authentication Credentials: Static
Default Username: wsitUser
Default Password: ••••••••
SAML Callback Handler: Browse...
Timestamp Timeout (s): 300
Key Obtention Iterations: 1000

Figure 8


localhost:8080/AuctionAppWebServiceClient/faces/index.xhtm
List
1..4/4

Id	Amount	Biddername	ItemId	
10	80000.0	Carolis	1	View Edit Destroy
2	10.0	Fali	1	View Edit Destroy
5	800.0	Maroc	4	View Edit Destroy
3	12.0	Vals	2	View Edit Destroy

Create New Bid
Index

Figure 6

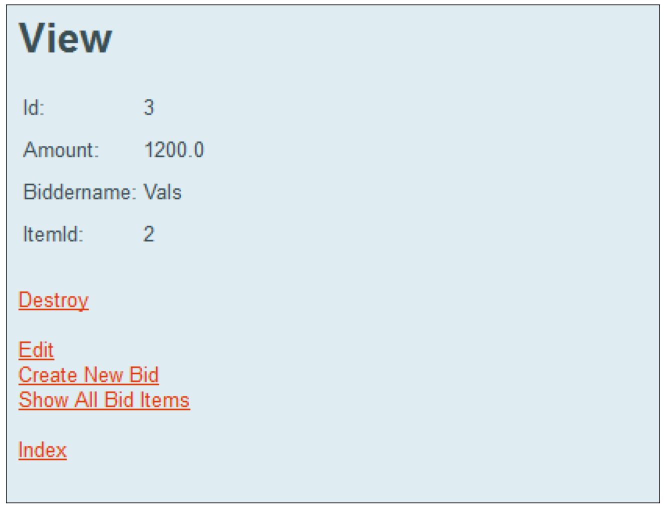

View
Id: 3
Amount: 1200.0
Biddername: Vals
ItemId: 2
Destroy
Edit
Create New Bid
Show All Bid Items
Index

Figure 9

2. Use the certificates to invoke the secured Web service to display the extrapolated amount of the bid:
   a. Make sure the AuctionApp we opened previously is up and running. If it is not, right-click the **AuctionApp** node and choose **Deploy**.
   b. Expand the **Web Service References** node of the AuctionAppWebService-Client project and right-click the **AuctionApp-SOAPws** node; then select **Edit Web Service Attributes**.
   c. Under the Quality Of Service tab, expand the **Security** section.
   d. Select the **Use development defaults** option, as shown in **Figure 8**.
   e. Click **OK**.
   f. Right-click the **AuctionApp-WebServiceClient** project and choose **Clean and Build**.
   g. Right-click the **AuctionAppWebService-Client** project again and choose **Run**.

   The list of all entries is displayed, as shown in **Figure 5**.
   h. Click the **Show all Bid Items** link to display the list of bid entries, as shown in **Figure 6**.
   i. Click the **View** link for the bidder named Vals to see the newly extrapolated amount of the Vals bid, as shown in **Figure 9**.

As you can see, the amount of the bid changed from 12.0 to 1200.0. This means that the client was able to call the secured Web service. Bravo!

Conclusion

In this article, we have seen how easy it is to configure security and reliability into an existing Web service and set up default credentials to access the service. The bundled GlassFish keystore and truststore files were very useful. We used NetBeans and Metro to automatically update these files and use them immediately for development.

In Part 2 of this series, we will focus on importing specific certificates into the GlassFish keystore and truststore so that they can be used in a production environment. </article>

LEARN MORE

- *NetBeans Advanced Web Service Interoperability manual*
- *Metro User Guide*
- GlassFish resources

JavaOne™

SEPT. 30 - OCT. 4 | SAN FRANCISCO

REGISTER NOW

Save $200 by Sept. 28th

Register at **oracle.com/javaone**

Silver Sponsor
JFrog

Bronze Sponsors
resin
CloudBees The Java™ PaaS Company
LIFERAY

ORACLE

GlassFish Monitoring with LightFish

A Conversation with Adam Bien

Java Champion **Adam Bien** discusses his new open source Java EE 6 stress test monitoring tool. **BY ARUN GUPTA**

Adam Bien is no stranger to Java developers, especially those of the Java EE stripe. He is a celebrated Java Champion and a JavaOne Rock Star who has given highly rated sessions at JavaOne, and he was named 2010 Java Developer of the Year by Oracle Magazine. *He works as a consultant in his native Germany (and throughout the continent) and is the author of several books on Java. We caught up with him to learn about LightFish, an open source monitoring tool for GlassFish.*

PHOTOGRAPHY BY
PIOTR MALECKI/GETTY IMAGES

Java Magazine: You recently released LightFish—a GlassFish monitoring tool. Tell us about it.

Bien: Stress tests are widely underestimated. In my client projects, I started to write simple tools to gather and analyze performance statistics during stress tests. For the *Java Magazine* article "Stress Testing Java EE 6 Applications," I wrote such a tool from scratch. During the 2011 JavaOne conference in San Francisco, an Oracle engineer asked me to participate in his talk and demonstrate the tool in action. After the session I got the idea of open sourcing the tool, which was named STM at the time. In fact, the first commit happened during the JavaOne conference and is still visible in the git logs.

LightFish is a Java EE 6 application that can be deployed on any Java EE 6 application server you like. LightFish accesses the "GlassFish under Stress" machine (see **Figure 1**) remotely via REST, gathers and persists the monitoring data, and re-exposes the data via REST again. LightView, the real-time monitoring client, always accesses the LightFish instance and never the GlassFish under Stress machine.

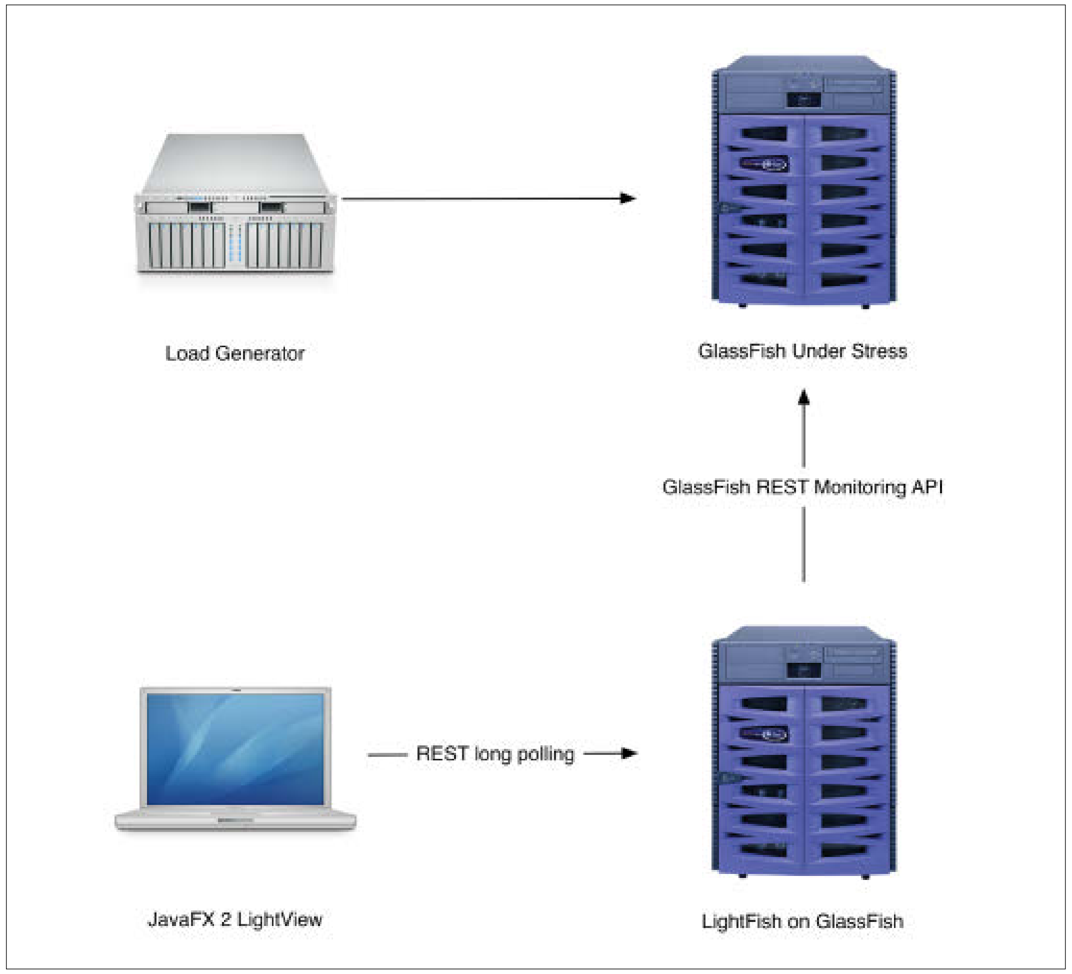
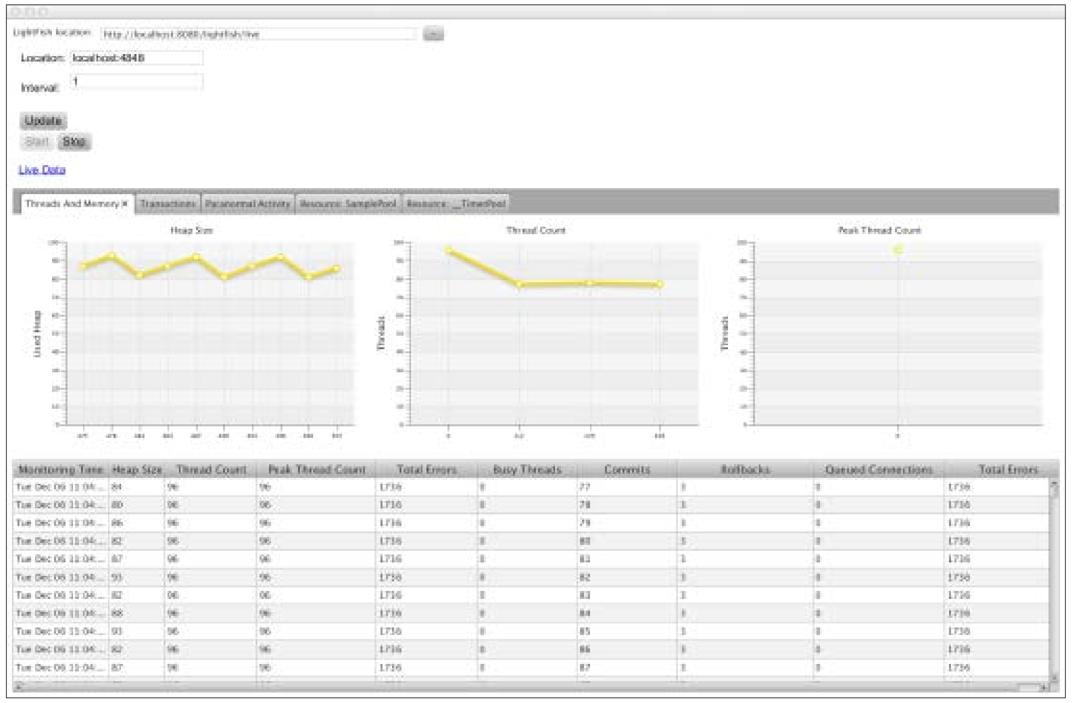


Figure 1



Monitoring Time	Heap Size	Thread Count	Peak Thread Count	Total Errors	Busy Threads	Commits	Rollbacks	Queued Connections	Total Errors
Tue Dec 06 11:04:...	84	96	96	1716	0	77	1	0	1716
Tue Dec 06 11:04:...	80	96	96	1716	0	78	1	0	1716
Tue Dec 06 11:04:...	86	96	96	1716	0	79	1	0	1716
Tue Dec 06 11:04:...	82	96	96	1716	0	80	1	0	1716
Tue Dec 06 11:04:...	87	96	96	1716	0	81	1	0	1716
Tue Dec 06 11:04:...	93	96	96	1716	0	82	1	0	1716
Tue Dec 06 11:04:...	82	96	96	1716	0	83	1	0	1716
Tue Dec 06 11:04:...	88	96	96	1716	0	84	1	0	1716
Tue Dec 06 11:04:...	93	96	96	1716	0	85	1	0	1716
Tue Dec 06 11:04:...	82	96	96	1716	0	86	1	0	1716
Tue Dec 06 11:04:...	87	96	96	1716	0	87	1	0	1716

Figure 2

Java Magazine: Talk about local deployment for development time monitoring and remote monitoring for production usage. What kind of telemetry data can be monitored?

Bien: Everything that GlassFish exposes. The GlassFish REST monitoring API exposes the data in a consistent way. Picking the right attribute is a matter of performing a single method invocation. Currently, LightFish monitors only the essential data: monitoringTime, usedHeapSize, threadCount, peakThreadCount, totalErrors, currentThread-Busy, committedTX, rolledBackTX, queuedConnections, activeSessions, expiredSessions, and escalationReason, as well as the monitoring data of all installed JDBC connections, the list of all installed applications, and the uptime (see **Figure 2**). These attributes should be self-descriptive. They are the attributes of the JPA [Java Persistence API] Snapshot entity, columns in the database, as well as the attributes exposed as an XML entity with the same structure. The whole metadata is derived from a single entity.

The JavaFX 2.0 client calls LightFish via a GET HTTP request. LightFish blocks the connection and waits until the availability of the next Snapshot initiated by the EJB timer expiration. The user gets the illusion of real-time monitoring without overloading the server. LightFish uses asynchronous Servlets 3.0 to implement this functionality and can actually handle several thousand monitoring clients. The JavaFX 2.0 application was built according to the MVC [model-view-controller] pattern with data binding. Each chart is bound to an attribute of the Snapshot entity. Because the monitoring data is persisted, you can use any reporting tool you like to analyze the data after the tests.

Java Magazine: Does LightFish provide an extensible architecture that allows users to expose their own monitoring data?

Bien: The recent version of LightFish supports scripting. You can register persistently at runtime new rules written in JavaScript, which are evaluated against the recent and the current Snapshot instances. You can easily express something like "please notify me if the heap size increase reaches a certain size." In this case, LightFish will create a new "Escalation" channel for you at runtime. You can subscribe to it with any tool you like. And the JavaFX 2.0 client, LightView, creates a new tab and displays all the escalated values in a table.

SIMPLE RULES

"Premature optimization is the root of all evil" is particularly true for Java EE 6. Write simple business code, stress test often, and optimize on demand.

The structure is hardcoded in the Snapshot entity. To analyze metrics that have not yet been exposed, you have to extend the Snapshot and LightView. The addition of a monitoring attribute usually takes about 15 minutes with testing.

However, I'm thinking about implementing a plug-in system.

Java Magazine: Suppose I'm building Java EE 6 applications on GlassFish. How can I benefit from LightFish?

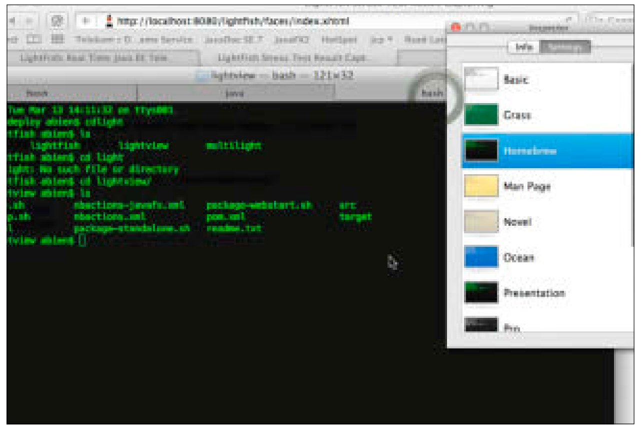
Bien: As a Java EE 6 developer, you have the unique opportunity to fully focus on the realization of business logic without thinking too much about framework libraries and

Java Champion Adam Bien and Java Evangelist Arun Gupta discuss LightFish while getting a caffeine fix. Both were in Poznań, Poland, for the GeeCON 2012 conference in May.

the infrastructure. LightFish helps to identify potential bottlenecks during nightly stress tests. "Premature optimization is the root of all evil" is particularly true for Java EE 6. Write simple business code, stress test often, and optimize on demand.

Java Magazine: What GlassFish features were used to build this application?

Bien: LightFish relies exclusively on GlassFish's management and monitoring REST APIs. LightFish uses the REST management API to enable monitoring and the REST monitoring API to fetch the monitoring data. There is no binary dependency on GlassFish. You could easily port LightFish to other application servers. The REST interface is vendor-specific, so you will have to reimplement a single method—org.lightfish.business.monitoring.control.SnapshotProvider#fetchSnapshot—to be able to monitor, for example, JBoss. Hence, I'm using GlassFish in the majority of my projects. I do not plan to reimplement the method yet.



```
http://localhost:8080/lightfish/faces/index.xhtml
lightview — bash — 121x32
Tue Mar 13 14:11:32 on ttys001
deploy abien$ cd light
tfish abien$ ls
    lightfish       lightview       multilight
tfish abien$ cd light
light: No such file or directory
tfish abien$ cd lightview/
tview abien$ ls
.sh        nbactions-javafx.xml    package-webstart.sh    src
p.sh       nbactions.xml           pom.xml                target
l          package-standalone.sh   readme.txt
tview abien$ 
```

Adam Bien demonstrates LightFish.

Java Magazine: Is there any performance overhead?

Bien: LightFish polls GlassFish. In the default case, LightFish will execute several GET requests every two seconds. This is just a default, and it is configurable at runtime. During nightly stress tests, you can use a longer interval and minimize the impact. LightFish continuously monitors my x-ray blog statistics software with negligible overhead.

Java Magazine: Why did you choose GlassFish to be the first app server? And, do you plan to port this application to other app servers?

Bien: My clients have been using GlassFish since version 2.0. I use GlassFish because all my Java EE projects are based on GlassFish. Porting LightFish to another application server is easy, but it is a rather boring task. I would prefer to implement new functionality instead. LightFish is my leisure-time open source project and, thus, it should be fun.

Java Magazine: What are your future plans, and how can the Java EE 6 community help improve LightFish?

Bien: There's a lot to do. Right now LightFish monitors only Java EE–specific behavior. I would like to dig into EJB [Enterprise JavaBeans] and JPA metadata as well. I'm also thinking of analyzing any application-dependent monitoring information. If you have an idea, just fork LightFish at github.com/AdamBien/lightfish. I have only one requirement: the code should be developed in the Java EE way—as simply as possible with minimal (none would be best) external dependencies. LightFish is developed under the Apache license.

Java Magazine: How can developers get started?

Bien: Download lightfish.war and drop it into the glassfish/domains/domain1/autodeploy folder. LightFish should install in a few seconds. No further interaction is needed. Then I would download the code. I tried to write simple code without any bloat. I even used LightFish code at recent conferences to explain Java EE capabilities. I've also recorded an introductory screencast [click the play button, left, to watch]. `</article>`

Arun Gupta (@arungupta) is a Java evangelist at Oracle, where he works to create and foster the community around Java EE, GlassFish, and Oracle WebLogic Server. He has more than 15 years of experience in the software industry.

LEARN MORE

- Adam Bien's blog

DEVELOPER POWER

With a bevy of Web-based development tools available, it's a great time to be a Java developer.

BY STEVE MELOAN
WITH CONTRIBUTIONS BY JANICE J. HEISS

Modern era agile software development is predicated upon iterative, incremental development processes, with continuous integration (CI) of fixes and enhancements. With the need for such CI comes the complexity of managing geographically distributed teams that are utilizing open source modules and frameworks obtained in a sometimes unregulated and undocumented fashion across the internet.

Fortunately, a rich stack of tool offerings enables development teams to more effectively track, build, integrate, and manage software projects and their modules. This article discusses many of the tools available today; however, this is by no means a complete, definitive list. You may have other favorites that are not included; let us know.

Java.net, among other things, provides a delivery channel for some of the technologies described in the following pages.

PHOTOGRAPHY BY BOB ADLER

EXPERT OPINION

"I find **Apache Ant** (Another Neat Tool) very useful for automating the build process. I especially appreciate its use of XML, which facilitates build-script creation, and the JUnit task, which facilitates integration of JUnit testing into the process."
—**Jeff Friesen**,
Freelance Developer and Educator

"I love using **Maven** in large organizations—where many developers are working on multiple projects. Some very deep thought about the software project lifecycle has gone into Maven over the years, and its strict standards approach works brilliantly where chaos can potentially abound. It doesn't matter which project developers join. A Maven `mvn clean install` command builds, tests, packages, and installs a local copy of the application for them, letting them get started with their coding.

Having good build and CI processes in place means that you can write code more quickly and maintain a higher bar of quality. In conjunction with TDD [test-driven development], build and CI means you can rapidly refactor without fear! Think of it as having a mentor looking over your shoulder—providing a safe and controlled environment in which you can quickly code and make bold changes."
—**Martijn Verburg**, Coauthor,
The Well-Grounded Java Developer

"For me, the real difference is between tools that have a point of view about how development processes should work and those that don't. I call these two types 'tools and metatools.' **Maven** is a build tool, because it essentially forces you to adapt to the Maven lifecycle and way of doing things. Ant doesn't come with such a point of view—instead, you have to build up targets and a sequence based on your own project lifecycle. Some teams prefer the additional structure which comes with a tool like Maven, whereas other teams object to having to fit in with the demands of the Maven lifecycle."
—**Ben Evans**, Coauthor,
The Well-Grounded Java Developer

"Identifying and downloading the correct versions of a Java project's dependent JAR files, which may have been obtained from a variety of online sources, is a thankless and often error-prone activity. **Maven** takes care of this for you, and effectively structures your project so as to eliminate a host of setup and configuration issues. Maven provides plug-ins that encourage and promote good standard software practices—including unit testing, version control, and standardized release processes.

A controlled process like Maven's makes development within a team far more efficient and scalable. And the documentation and site reports add to Maven's value as a full-featured project management and comprehension tool."
—**Cas Saternos**, Oracle Certified DBA and Sun Certified Java Programmer

01 BUILD AUTOMATION TOOLS

A build automation tool ideally provides the ability to build a given project with a single command, mobilizing all the modules, artifacts, libraries, and code necessary to that project.

Apache Maven. Hosted by the Apache Software Foundation and written in Java, Maven can be used to build projects in Java, as well as C#, Ruby, Scala, and other languages. It operates from an XML file, but uses a very different paradigm from Ant. Rather than simply chaining together sequential build tasks, as occurs with Ant, Maven defines a project in terms of its dependencies, external modules and components, build order, directories, and required plug-ins. Maven projects are defined via a Project Object Model (POM) file (pom.xml). Maven dynamically downloads Java libraries and Maven plug-ins from either the Maven Central repository or other defined software repositories.

Larger Maven projects are typically divided into several subprojects, each with its own POM file, but with a root POM to compile the master project via a single command. The Maven plug-in architecture allows it to interface with build tools for other languages, including the .NET framework and C/C++. Popular IDEs that support development with Maven include Eclipse, IntelliJ, JBuilder, JDeveloper, and NetBeans.

Gradle. Written in Java and Groovy, Gradle builds upon the concepts of Ant and Maven but uses a Groovy-based Domain-

Ant/Ivy 101

Apache Ant. Similar to the decades-old UNIX Make in some respects, Apache Ant is written in Java and is best suited to building Java projects. It uses an XML file (build.xml) to define a given build process and its dependencies. Within the build file, Ant can also delegate build work to either native or Java-based external programs.

One of Ant's primary goals was to solve the portability issues of Make, where different platforms required different script commands. Ant provides built-in functionality designed to behave the same on all platforms. It has limited fault-handling capabilities and no persistence of state, so it is primarily useful only for classic build and test processes.

Ant is supported by most major IDEs, including Eclipse, IntelliJ, JBuilder, JDeveloper, NetBeans, and WebSphere.

Apache Ivy. Written in Java, Apache Ivy is a subproject of the Apache Ant project, serving as a transitive relation dependency manager. An XML file defines project dependencies and the resources necessary to build the given project. Ivy resolves and downloads required resources from the specified repositories. Whereas Maven is a complete build tool, with built-in dependency management, Ivy focuses specifically on dependency management functionality, working in partnership with Ant.

EXPERT OPINION

"**Hudson/Jenkins** are CI tools, so the first thing that comes to mind is that they 'execute tests.' Correct. But they do so much more. With the proper set of plug-ins—such as test coverage and static analysis—they provide everyone in the development team with a highly valuable information center on the health of the project. Developers can autonomously verify the quality of the code they are writing, and project leaders can keep everything under control, monitoring both the progress and the technical debt. The best of Hudson/Jenkins happens with Maven-based projects, thanks to Maven profiles. This combination makes it easier to build on multiple configurations—such as JDKs and databases. In addition, Maven can warn you about whether the library versions you're using will break your build due to a regression."
—**Fabrizio Giudici**,
Senior Java Architect, Tidalwave

"My metatool of choice is NetBeans, Java EE Edition. It integrates **Maven 3; JUnit; Ant; Hudson/Jenkins;** and countless Java EE hints, wizards, and extensions. You get an extremely productive environment with a single click—no plug-in fiddling required."
—**Adam Bien**, Java Champion

"Combining smart features of both Ant and Maven, and being powered by Groovy, **Gradle** provides a new and powerful way to handle your delivery pipelines."
—**Michael Hüttermann**, Java Champion

"Having been a Java developer since nearly the beginning (summer of '95), I have a collection of open source projects that I either founded or still maintain. As a full-time researcher, and also a new father, I need a system to reliably manage these projects and push out releases. That's what **Hudson** is to me. I've been using it for several years now and continue to love it—it just works."
—**Josh Marinacci**, Java Champion

"**Jenkins** is an awesome tool for continuous integration. With it, you can combine all the work of a team with less error and more quality/productivity—while monitoring status and the number of tests. And, yes, you can use JUnit! There are almost too many available plug-ins—for example, Sonar, which verifies the quality of your team's code, test coverage, duplication of code, and so on."
—**Otávio Gonçalves de Santana**,
JUG Leader, Java Bahia

Specific Language (DSL) rather than XML. Gradle uses a directed acyclic graph (DAG) to determine the order in which build tasks should be run. Gradle's DSL is extensible, allowing for the addition of new language elements or the enhancement of existing elements. Intended to manage large, multiproject builds, Gradle intelligently determines which part of a build tree is current, so that already up-to-date dependent branches needn't be rebuilt.

Gradle offers support and transitive dependency management for existing Maven and Ivy repositories and also provides a converter to translate Maven POM files into Gradle scripts.

02 CONTINUOUS INTEGRATION TOOLS

When properly implemented, CI requires that each commit of new software be accompanied by a complete build and run, and that it pass all defined unit tests. With the advent of CI tools, this change in commit functionality has increasingly become highly sophisticated and automated. CI tools include CruiseControl, Hudson, Jenkins, Bamboo, BuildMaster, AnthillPro, and Teamcity.

The big-picture goal of such CI tools is to wrap configurable intelligence (that can be extended with plug-ins) around the process of version control, builds, testing, and reporting of results. Below is a sampling of popular CI tools, including two recent winners of the Duke's Choice Award, Hudson and Jenkins.

Hudson. Winner of the 2008 Duke's Choice Award in the Developer Solutions category, Hudson is a popular alternative to CruiseControl. Hudson provides an easy-to-use, GUI-based configurable system for integrating changes to a project—obtaining explicit fresh builds of the project, scheduling future builds, and monitoring the results of externally run jobs (such as cron jobs), including those that execute on remote machines. Results can be monitored via e-mail or RSS, and third-party plug-ins offer additional extensible functionality.

Hudson is written in Java, and runs in a Servlet container (such as Apache Tomcat or GlassFish). It can execute Ant- and Maven-based projects as well as simple shell scripts and Microsoft Windows batch commands, and can distribute build/test loads to multiple computers, as well as keep track of which builds produced which JARs. Plug-ins provide integration with most version control systems and bug databases and can add new functionality or even change the appearance of Hudson. Meanwhile, build test reports can be generated in a variety of formats (JUnit is supported out of the box).

Oracle continues to develop Hudson along with the community at large. But in January 2011, a fork of the project was created and named Jenkins.

Jenkins. Originally begun as the Hudson

CI tool, the Jenkins project was created in January 2011. Both Hudson and Jenkins consider the other to be a fork, with separate development branches.

Winner of the 2008 Duke's Choice Award (as Hudson) in the Developer Solutions category, Jenkins provides an easy-to-use, GUI-based configurable system for integrating changes to a project, obtaining explicit fresh builds of a project, scheduling future builds, and monitoring the results of externally run jobs (such as cron jobs), including those that execute on remote machines. Results can be monitored via e-mail or RSS, and third-party plug-ins offer additional extensible functionality.

Jenkins is written in Java, and runs in a Servlet container (such as Tomcat or GlassFish). It can execute Ant- and Maven-based projects as well as simple shell scripts and Windows batch commands, and can distribute build/test loads to multiple computers, as well as keep track of which builds produced which JARs. Plug-ins provide integration with most version control systems and bug databases and can add new functionality or even change the appearance of Jenkins. Meanwhile, build test reports can be generated in a variety of formats (JUnit is supported out of the box).

CruiseControl. Written in Java, CruiseControl provides an extensible framework for custom CI processing. Its features include a Web interface to monitor current and past builds; plug-ins for a variety of source controls, build technologies, and notification schemes; interfaces for popular build automation tools such as Ant and Maven, as well as a standard exec builder; and ports of CruiseControl for .NET (CruiseControl.NET) and Ruby (CruiseControl.rb).

ToolsCloud. Almost in a conceptual category by itself, ToolsCloud is effectively a hybrid cloud-based IDE that includes project management, CI functionality (including build automation and automated testing), metrics and analysis, and a broad array of development tools. Available via a monthly subscription plan, and hosted on Amazon Elastic Compute Cloud (EC2), ToolsCloud includes Git (software configuration management), Redmine (project management and bug-tracking tool), Nexus (artifact management), Hudson/Jenkins (CI), and more. ToolsCloud features a management calendar and tracking, reporting, and statistics tools as well as metrics history and analysis tools. It easily integrates with your IDE of choice.

EXPERT OPINION

"Using **Jenkins** is like having an extra member in your development team. It relentlessly builds and tests your software, preventing mistakes and freeing up time so that your team can focus on simply producing great software."
—**Juliano Viana**,
Founder and CTO of LogicStyle

03 OPEN SOURCE MODULES

Because modern software development involves geographically distributed teams that use diverse open source modules and frameworks acquired from across the internet, projects can run considerable security, consistency, and reliability risks. According to a survey conducted by Sonatype, more than 80 percent of typical software applications consist of open source components and frameworks. Yet many studies find a staggering use of vulnerable, insecure, and nondocumented open source offerings. Sonatype finds that only 32 percent of teams maintain a detailed "bill of materials" record of the open source components in their development stack. The survey also revealed that only 50 percent of developers report that their company has an open source software policy.

Code/artifact repository. In order to deal with an ever growing and ever more complex open source ecosystem, enterprises are increasingly turning to code repositories and cached repository management systems—establishing a centralized, secure, and managed code repository for the artifacts required to build and maintain projects.

A code repository establishes a platform for the storage, retrieval, and management of the binary software artifacts and metadata necessary for a given project or application. The information is archived and organized in such a way that build tools such as Maven or Ant/Ivy can effectively locate and process the information.

In the case of a typical Maven repository, the binary artifact is a JAR file, but it could just as easily be a Flash library or a Ruby library. When a Maven POM file lists a project dependency that includes a repository-based entry, it downloads that entry's POM, and then downloads any libraries or modules required by that dependency. This ability to determine a project's dependencies and transitive dependencies is made possible by the standards and structure of the repository.

Sonatype/Central repository. The default configuration of Maven retrieves software artifacts from the Central repository, a public facility that is stewarded by Sonatype. Central reportedly receives four billion requests per year, contains 300,000 components, and is accessed by 60,000 development organizations worldwide. The average enterprise reportedly downloads more than 1,000 unique components from Central each month.

Repositories such as Central and tools such as Maven typically use a Group, Artifact, Version (GAV) "coordinate" system as a means of storing and locating a given artifact: http://repo1.maven.org/groupID/artifactID/version/.

For example, components produced by the Maven project at the Apache Software Foundation would be stored/located under a groupID of org.apache.maven. An artifactID is the identifier for a given component. The combination of groupID and artifactID uniquely identifies a given project, and the version identifier specifies the version of the project, while the packaging identifier specifies the binary software format.

Once an artifact is assigned a release number on Central, the file contents cannot be altered. The Central repository also contains cryptographic hashes and PGP signatures that can be used to verify artifact authenticity and integrity.

While Maven can be configured to retrieve software artifacts directly from one of the many Central mirror sites around the world (or any external repository), an increasingly popular option that is faster, more secure, and more easily managed is to employ a repository manager as a locally controlled proxy to Central and other artifact repositories (such as those provided by Oracle, Red Hat, and Codehaus).

04 SOFTWARE CONFIG MANAGEMENT

Software configuration management (SCM) entails rigorously controlling and tracking changes made to software, and includes a subfunctionality of revision control (version handling). Development tools such as Hudson and Jenkins (explored earlier) offer support for a variety of such SCM tools, including Clearcase, CVS, Git, and Subversion.

Git. Git offers a distributed revision control and SCM system, suitable to handle both large and small development projects. GitHub provides a collaborative, Web-based facility to manage both public and private Git repositories. Written using Ruby on Rails, GitHub is the most popular Git hosting site, providing social networking functionality and usage data specifically directed toward collaborative development.

EXPERT OPINION

"**Nexus** is a rock-solid vault for your binaries, tailor-made for a Maven-based build process."

—**John Ferguson Smart**, CEO, Wakaleo Consulting

"**Artifactory** is a great choice for DevOps. Its integration with Jenkins offers full traceability across builds, links back to tickets, and allows comfortable build promotions. Easy configuration, openness, and extensibility make Jenkins a central service hub and a smart backbone of your continuous delivery and DevOps infrastructure."

—**Michael Hüttermann**, Java Champion

05 REPOSITORY MANAGEMENT

A repository management system offers a locally cached proxy between development teams and external repositories. It speeds download times, ensures managed and configurable access to both external artifacts and internally created modules, and provides tagged and searchable metadata.

Sonatype Nexus. Nexus is a managed, central point of access for external repositories, offering configurable permissions and customizable/searchable user-defined metadata. **Figure 1** depicts how a repository manager fits into a typical development process.

Nexus provides a centralized point for managed access of open source software components and their dependencies, serving as a configurable proxy between organizational and public repositories. Nexus offers cached components for quick download, ensures that all users access the same modules, enables secure and controllable deployment of internally developed components, and provides configurable, partner-specific access. Meanwhile, user-defined metadata offers rich and customizable search capabilities.

JFrog Artifactory. Winner of the 2011 Duke's Choice Award for Innovative Tool for Developers, JFrog Artifactory is a Java-based binary file repository management tool, with a free open source version, a paid Pro version, and a software-as-a-service (SaaS) cloud-based version (Artifactory Online). **Figure 2** shows how Artifactory acts as a proxy between your Maven client and the outside world.

JFrog Artifactory serves as a proxy between build tools such as Maven, Ant,

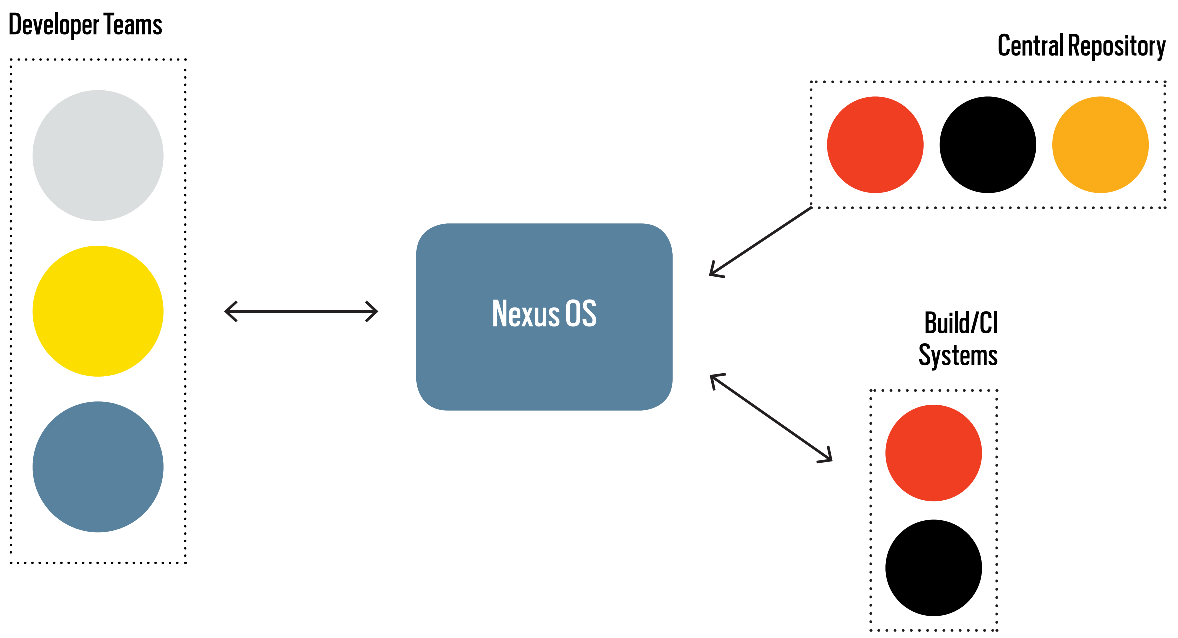


Figure 1

Ivy, Gradle, and so on—providing local, fast-access caching of remote artifacts, offering configurable management of repository access permissions and customizable/searchable user-defined metadata. Artifactory is built on top of the Java Content Repository (JCR), packaged as a standard Java EE Web application, and deployable into any standard Servlet container (Tomcat, WebSphere, JBoss, GlassFish, and so on.)

The Jenkins CI Artifactory Online repository service—a Jenkins/JFrog collaboration offering a cloud-based Artifactory repository solution developed specifically for the Jenkins community—was announced at the April 2012 Jenkins User Conference in Paris, France.

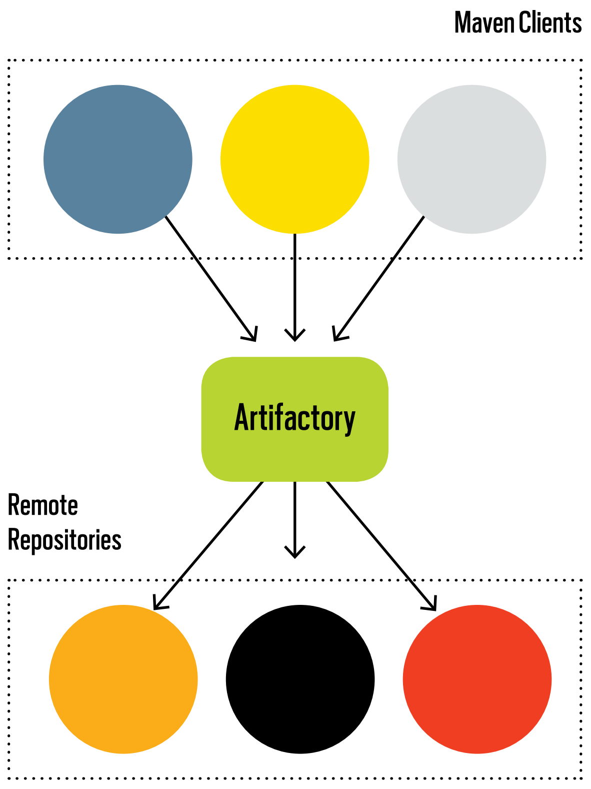


Figure 2

06 RUNTIME ANALYSIS

While not explicitly directed at collaborative software development/management, runtime analysis tools offer an essential debugging/tuning window into applications during actual execution and are, therefore, an often-used tool class in the development lifecycle.

VisualVM. VisualVM provides an intuitive, graphical interface that allows developers to monitor and troubleshoot executing Java applications. While VisualVM itself requires JDK 6 to run, it can monitor any application running at JDK 1.4.2 or greater. Utilizing such technologies as jvmstat, Java Management Extensions (JMX), the Serviceability Agent (SA), and the Attach API, VisualVM displays both local and remote applications, offering visual data on CPU usage, Garbage Collection (GC) activity, heap and permanent generation memory, loaded classes, running threads, and more. VisualVM also allows for offline analysis of core dumps, as well as analysis of taken thread dumps, heap dumps, and profiler snapshots. And because VisualVM is built on the NetBeans platform, it is readily extensible with plug-ins (available at the VisualVM Plugin Center). Finally, VisualVM can be integrated with such IDEs as Eclipse and NetBeans.

JRebel. Developed by Jevgeni Kabanov and Toomas Römer of ZeroTurnaround, JRebel is a plug-in for the Java Virtual Machine (JVM) that enables instant reloading of changes made to a Java class file. In 2011, JRebel won the JAX Innovation Award for Most Innovative Java Technology. At JavaOne 2011, JRebel was awarded the Duke's Choice Award for Innovative Compiler for Java Code. Java-based and usable on any operating system that supports Java, JRebel is IDE agnostic and designed for integration with various Java EE standards and Java application servers. It is freely available to open source software projects and developers using Scala. It supports immediately visible code changes without redeploying; handles changes to class structures, frameworks, and Java EE; and supports all major Java application servers, IDEs, and frameworks. It eliminates memory leaks and build time during development and supports Apache Ant and Apache Maven. </article>

Steve Meloan is a former C/UNIX software developer who has covered the Web and the internet for such publications as *Wired*, *Rolling Stone*, *Playboy*, *SF Weekly*, and the *San Francisco Examiner*. He recently published a science-adventure novel, *The Shroud*, and regularly contributes to *The Huffington Post*.

Janice J. Heiss is the Java acquisitions editor at Oracle and a technology editor at *Java Magazine*.

EXPERT OPINION

"**VisualVM** is a tool that every Java developer should become familiar with. It comes as part of every modern Java SDK and enables developers to visualize the internals of their applications as they run, helping diagnose even the most-difficult performance problems. And the integration with BTrace means that you can perform complex inspections on running production systems, without affecting end-user experience."
—**Juliano Viana**,
Founder and CTO of LogicStyle

"I think **VisualVM** is underrated by many Java developers. Not only does it enable you to monitor memory consumption and threads, it's invaluable when it comes to memory analysis. With this tool, I've easily found the causes of a number of memory leaks in customer projects."
—**Fabrizio Giudici**,
Senior Java Architect, Tidalwave

POONAM BAJAJ

BIO

HotSpot's Hidden Treasure

The HotSpot Serviceability Agent's powerful tools can debug live Java processes and core files.

The HotSpot Serviceability Agent is a hidden treasure present in the JDK that very few people know about. The Serviceability Agent (SA) is a set of Java APIs and tools that can be used to debug live Java processes and core files (also called *crash dumps* on Microsoft Windows).

SA can examine Java processes or core files, which makes it suitable for debugging Java programs as well as the Java HotSpot VM. It is a snapshot debugger and lets us look at the state of a frozen Java process or a core file. When SA is attached to a Java process, it stops the process at that point and we can explore the Java heap; look at the threads that were running in the process at that point; examine internal data structures of the Java HotSpot VM; and look at the loaded classes, compiled code of methods, and so on. The process resumes after SA is detached from it.

SA Binaries

Before we go into the details about the features and utilities that SA offers, I would like to mention SA binaries that are present in the JDK. There are two SA binaries that are shipped with the JDK:

- For Microsoft Windows: sa-jdi.jar and jvm.dll
- For Oracle Solaris and Linux: sa-jdi.jar and libsaproc.so

These binaries provide the SA Java APIs and also include useful debugging tools implemented using these APIs.

JDK Versions with Complete SA Binaries

The following JDK versions have complete SA binaries:

- JDK 7 on all platforms
- 6u17+ on Oracle Solaris and Linux
- 6u31+ on Microsoft Windows

Prior to these versions, SA was not shipped with the JDK on Microsoft Windows, and only a subset of SA classes was shipped with JDKs on Oracle Solaris and Linux. The JDK versions above make the complete set of SA classes available on all of these platforms.

Why Use SA?

Why use SA when we have native debugging tools such as dbx, GDB, WinDbg, and many others?

First, SA is a Java-based, platform-independent tool, so it can be used to debug Java processes and cores on all the platforms where Java is supported. Additionally, debugging a Java process or the Java HotSpot VM with native debuggers is very limiting, because although native debuggers can help us examine the native OS process state, they cannot help us examine the Java or the Java Virtual Machine (JVM) state of the process.

For example, if I need to view the objects in the Java heap, native debuggers would show me the raw hex numbers, whereas SA has the ability to interpret those hex numbers and present the object view instead. SA has knowledge about the Java heap, such as its boundaries, objects in the Java heap, loaded classes, thread objects, and internal representations of the Java HotSpot VM. SA makes it very easy for us to examine the Java-level details and JVM-level details of the Java process or core file.

DETAILS, DETAILS

SA makes it very easy to **examine the Java-level details and JVM-level details** of a Java process or core file.

SA Debugging Tools

There are two main SA debugging tools implemented using SA APIs:

- HSDB, which is a GUI tool and the main debugger
- CLHSDB, which is a command-line variant of HSDB

HSDB: The GUI debugger. HSDB facilitates examining Java processes, core files, and also remote Java processes. Let's see how we can launch and use it on a Microsoft Windows machine.

First, we need to set some environment variables. Set SA_JAVA to the location of the Java executable in the JDK/bin folder, for example:

```
set SA_JAVA=
d:\java\jdk1.7.0_03\bin\java
```

On Microsoft Windows, the PATH environment variable should contain the location of the JVM binary used by the target process or core and also the folder where the Debugging Tools for Windows are installed on the machine, for example:

```
set PATH= d:\java\jdk1.7.0_03\bin\
server;d:\windbg;%PATH%
```

Set the PATH environment variable and then launch HSDB as follows:

```
java -Dsun.jvm.hotspot.debugger.
useWindbgDebugger=true -classpath
d:\java\jdk1.7.0_03\lib\sa-jdi.jar
sun.jvm.hotspot.HSDB
```

On an Oracle Solaris or Linux machine, we just need to set SA_JAVA to the Java executable and then we can launch HSDB as follows:

```
java -Dsun.jvm.hotspot.debugger.
useProcDebugger=true -classpath
/java/jdk1.7.0/lib/sa-jdi.jar
sun.jvm.hotspot.HSDB
```

These launch commands bring up the HSDB GUI tool, as shown in **Figure 1**.

Let's take a quick look at some of the very useful utilities available in this tool, which are shown in **Figure 2**.

Figure 3 shows the Object Inspector, which you can use to inspect Java objects.

Figure 4 shows how you can find where a particular address lies in the Java process.

Figure 5 shows the Object Histogram.

You can find the heap boundaries, as shown in **Figure 6**.

CLHSDB: The command-line debugger. CLHSDB is the command-line variant of HSDB.

We need to set the same environment variables for CLHSDB as we did for HSDB. Use the following command to launch this tool on Microsoft Windows:

```
java -Dsun.jvm.hotspot.debugger.
useWindbgDebugger=true -classpath
d:\java\jdk1.7.0_03\lib\sa-jdi.jar
sun.jvm.hotspot.CLHSDB
```

CLHSDB offers almost all the features that the GUI version of the tool offers.

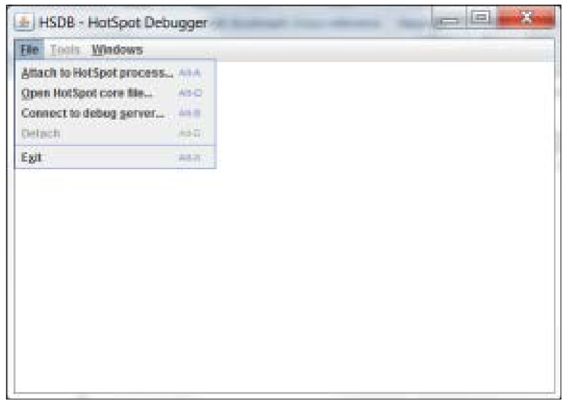
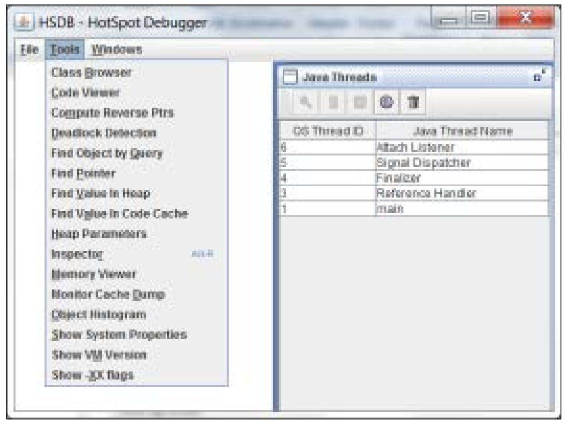
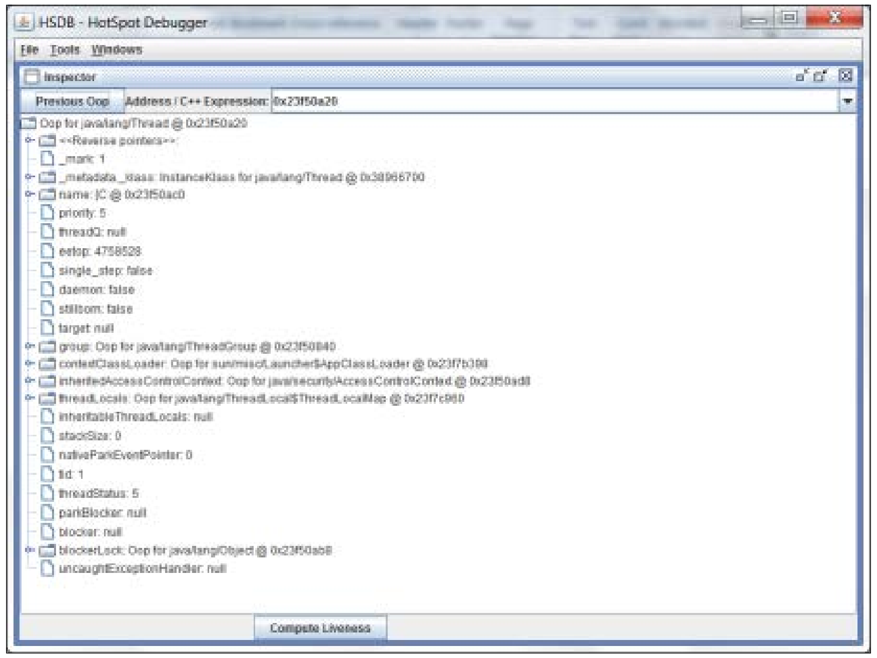

HSDB - HotSpot Debugger
File Tools Windows
Attach to HotSpot process... Alt-A
Open HotSpot core file... Alt-C
Connect to debug server... Alt-S
Detach Alt-D
Exit Alt-X

Figure 1



OS Thread ID	Java Thread Name
6	Attach Listener
5	Signal Dispatcher
4	Finalizer
3	Reference Handler
1	main

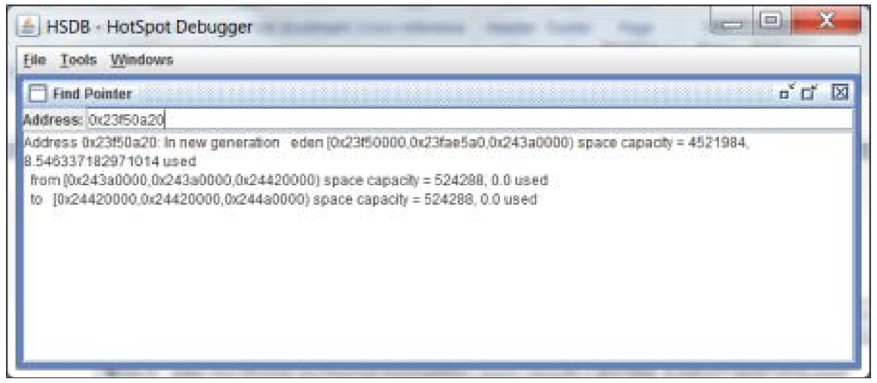
Figure 2



Address / C++ Expression: 0x23f50a20

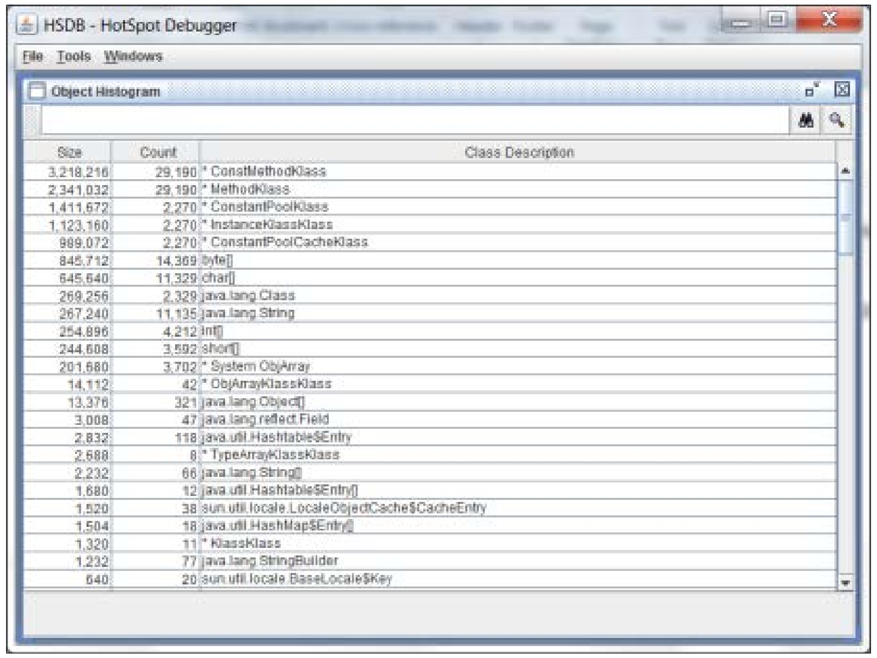
```
Oop for java/lang/Thread @ 0x23f50a20
  <<Reverse pointers>>
  _mark: 1
  _metadata._klass: InstanceKlass for java/lang/Thread @ 0x38966700
  name: [C @ 0x23f50ac0
  priority: 5
  threadQ: null
  eetop: 4758528
  single_step: false
  daemon: false
  stillborn: false
  target: null
  group: Oop for java/lang/ThreadGroup @ 0x23f50840
  contextClassLoader: Oop for sun/misc/Launcher$AppClassLoader @ 0x23f7b398
  inheritedAccessControlContext: Oop for java/security/AccessControlContext @ 0x23f50ad8
  threadLocals: Oop for java/lang/ThreadLocal$ThreadLocalMap @ 0x23f7c960
  inheritableThreadLocals: null
  stackSize: 0
  nativeParkEventPointer: 0
  tid: 1
  threadStatus: 5
  parkBlocker: null
  blocker: null
  blockerLock: Oop for java/lang/Object @ 0x23f50ab8
  uncaughtExceptionHandler: null
```

Figure 3



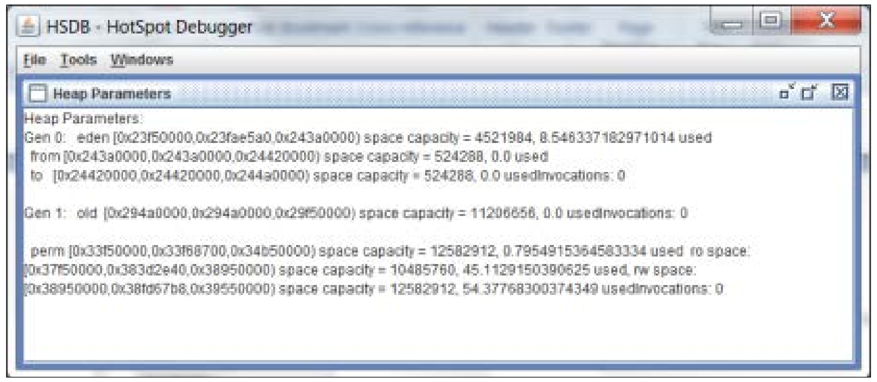
```
Address: 0x23f50a20
Address 0x23f50a20: In new generation   eden [0x23f50000,0x23fae5a0,0x243a0000) space capacity = 4521984,
8.546337182971014 used
 from [0x243a0000,0x243a0000,0x24420000) space capacity = 524288, 0.0 used
 to   [0x24420000,0x24420000,0x244a0000) space capacity = 524288, 0.0 used
```

Figure 4



Size	Count	Class Description
3,218,216	29,190	* ConstMethodKlass
2,341,032	29,190	* MethodKlass
1,411,672	2,270	* ConstantPoolKlass
1,123,160	2,270	* InstanceKlassKlass
989,072	2,270	* ConstantPoolCacheKlass
845,712	14,369	byte[]
645,640	11,329	char[]
269,256	2,329	java.lang.Class
267,240	11,135	java.lang.String
254,896	4,212	int[]
244,608	3,592	short[]
201,680	3,702	* System ObjArray
14,112	42	* ObjArrayKlassKlass
13,376	321	java.lang.Object[]
3,008	47	java.lang.reflect.Field
2,832	118	java.util.Hashtable$Entry
2,688	8	* TypeArrayKlassKlass
2,232	66	java.lang.String[]
1,680	12	java.util.Hashtable$Entry[]
1,520	38	sun.util.locale.LocaleObjectCache$CacheEntry
1,504	18	java.util.HashMap$Entry[]
1,320	11	* KlassKlass
1,232	77	java.lang.StringBuilder
640	20	sun.util.locale.BaseLocale$Key

Figure 5



```
Heap Parameters:
Gen 0:   eden [0x23f50000,0x23fae5a0,0x243a0000) space capacity = 4521984, 8.546337182971014 used
 from [0x243a0000,0x243a0000,0x24420000) space capacity = 524288, 0.0 used
 to   [0x24420000,0x24420000,0x244a0000) space capacity = 524288, 0.0 usedInvocations: 0

Gen 1:   old  [0x294a0000,0x294a0000,0x29f50000) space capacity = 11206656, 0.0 usedInvocations: 0

 perm [0x33f50000,0x33f68700,0x34b50000) space capacity = 12582912, 0.7954915364583334 used  ro space:
[0x37f50000,0x383d2e40,0x38950000) space capacity = 10485760, 45.1129150390625 used, rw space:
[0x38950000,0x38fd67b8,0x39550000) space capacity = 12582912, 54.37768300374349 usedInvocations: 0
```

Figure 6

For example, to examine any Java object, use the inspect command, as shown in **Listing 1**.

To look at heap boundaries, we can use the universe command, as shown in **Listing 2**.

Listing 3 and **Listing 4** show the complete list of commands available with this tool.

Other Tools

There are some other very handy small utilities bundled with SA. Let's see how to use them and how their output looks:

- FinalizerInfo prints details on the finalizable objects, as shown in **Listing 5**.
- HeapDumper dumps the heap in HPROF format, as shown in **Listing 6**.
- PermStat prints the permanent generation statistics, as shown in **Listing 7**.
- PMap prints the process map of the process (see **Listing 8**), much like the Oracle Solaris pmap tool does.
- SOQL, the Structured Object Query Language tool, is an SQL-like language that we can use to query the Java heap, as shown in **Listing 9**. JHat also provides an interface for using this language, and pretty good documentation on this language is also available in JHat.
- JSDB, the JavaScript Debugger, provides a JavaScript interface to SA (see **Listing 10**). It is a command-line JavaScript shell based on Mozilla's Rhino JavaScript engine. More details on this utility can be found in the open source Java HotSpot VM repository in the file hotspot/agent/doc/jsdb.html.

LISTING 1 / LISTING 2 / LISTING 3 / LISTING 4 / LISTING 5 / LISTING 6

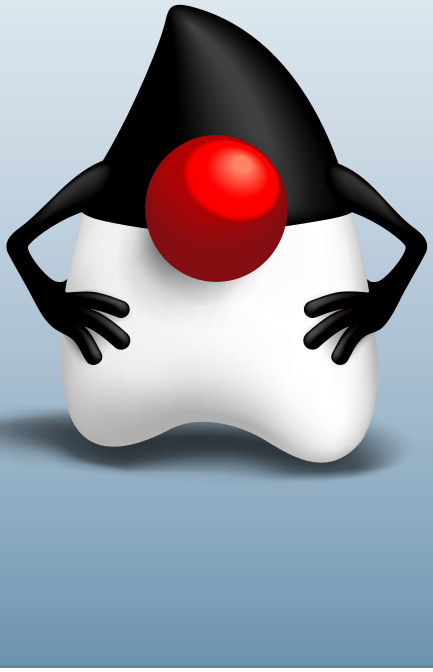
```
hsdb> inspect 0x23f50a20
instance of Oop for java/lang/Thread @ 0x23f50a20 @ 0x23f50a20 (size = 104)
_mark: 1
_metadata._klass: InstanceKlass for java/lang/Thread @ 0x38966700 Oop @
0x38966700
name: [C @ 0x23f50ac0 Oop for [C @ 0x23f50ac0
priority: 5
threadQ: null null
eetop: 4758528
single_step: false
daemon: false
stillborn: false
target: null null
group: Oop for java/lang/ThreadGroup @ 0x23f50840 Oop for java/lang/Thread-
Group @ 0x23f50840
contextClassLoader: Oop for sun/misc/Launcher$AppClassLoader @ 0x23f7b398 Oop
for sun/misc/Launcher$AppClassLoader @ 0x23f7b398
inheritedAccessControlContext: Oop for java/security/AccessControlContext @
0x23f50ad8 Oop for java/security/AccessControlContext @ 0x23f50ad8
threadLocals: Oop for java/lang/ThreadLocal$ThreadLocalMap @ 0x23f7c960 Oop
for java/lang/ThreadLocal$ThreadLocalMap @ 0x23f7c960
inheritableThreadLocals: null null
stackSize: 0
nativeParkEventPointer: 0
tid: 1
threadStatus: 5
parkBlocker: null null
blocker: null null
blockerLock: Oop for java/lang/Object @ 0x23f50ab8 Oop for java/lang/Object @
0x23f50ab8
uncaughtExceptionHandler: null nullCheck heap boundaries
```

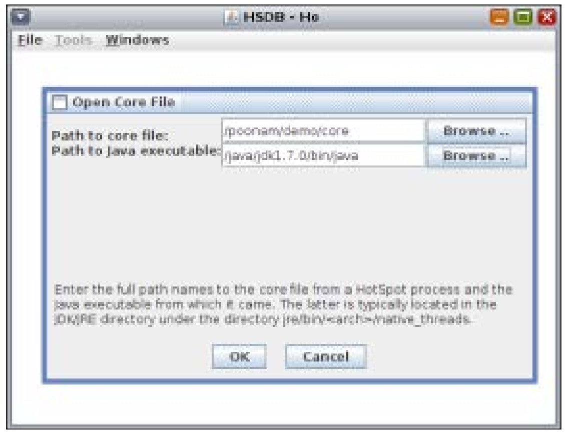
Download all listings in this issue as text

Let's Get Our Hands Dirty

Let's get a real feel for the SA tools and debug a Java program crash using them. I have a simple program of Java Native Interface (JNI) code that writes to a byte array beyond its size limit, which results in overwriting and corrupting the object that follows it in the Java heap. This causes the program to crash when the garbage collector tries to scan the heap. See **Listing 11**.

The crash happened in objArrayKlass::oop_follow_contents(oopDesc*) at program counter

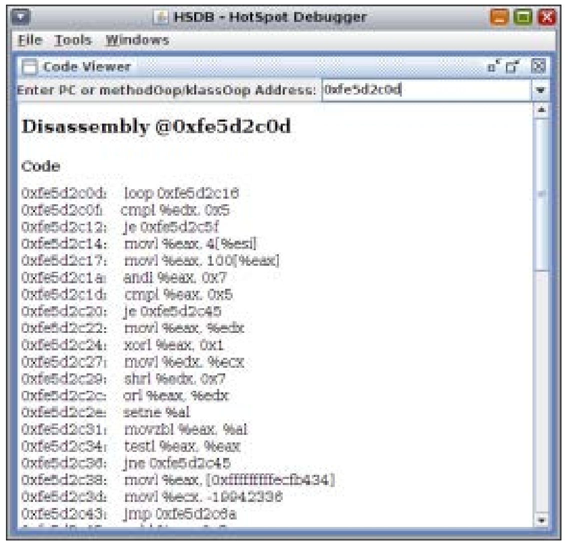




```
HSDB - Ho
File  Tools  Windows
Open Core File
Path to core file:          /poonam/demo/core          Browse ..
Path to java executable:    /java/jdk1.7.0/bin/java    Browse ..

Enter the full path names to the core file from a HotSpot process and the
Java executable from which it came. The latter is typically located in the
JDK/JRE directory under the directory jre/bin/<arch>/native_threads.
OK   Cancel
```

Figure 7



```
HSDB - HotSpot Debugger
File  Tools  Windows
Code Viewer
Enter PC or methodOop/klassOop Address: 0xfe5d2c0d

Disassembly @0xfe5d2c0d

Code

0xfe5d2c0d:   loop 0xfe5d2c16
0xfe5d2c0f:   cmpl %edx, 0x5
0xfe5d2c12:   je 0xfe5d2c5f
0xfe5d2c14:   movl %eax, 4[%esi]
0xfe5d2c17:   movl %eax, 100[%eax]
0xfe5d2c1a:   andl %eax, 0x7
0xfe5d2c1d:   cmpl %eax, 0x5
0xfe5d2c20:   je 0xfe5d2c45
0xfe5d2c22:   movl %eax, %edx
0xfe5d2c24:   xorl %eax, 0x1
0xfe5d2c27:   movl %edx, %ecx
0xfe5d2c29:   shrl %edx, 0x7
0xfe5d2c2c:   orl %eax, %edx
0xfe5d2c2e:   setne %al
0xfe5d2c31:   movzbl %eax, %al
0xfe5d2c34:   testl %eax, %eax
0xfe5d2c36:   jne 0xfe5d2c45
0xfe5d2c38:   movl %eax, [0xffffffffecfb434]
0xfe5d2c3d:   movl %ecx, -19942336
0xfe5d2c43:   jmp 0xfe5d2c8a
```

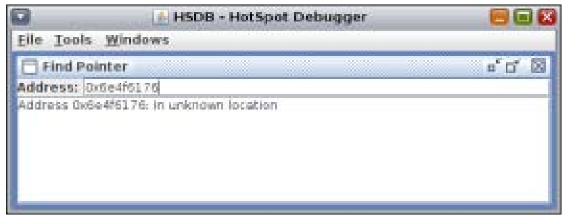
Figure 8

(PC) 0xfe5d2c17. See **Listing 12**, which shows the stack trace of the crash from the hs_err file.

With the crash, a core file got generated. Let's open this core with HSDB (see **Figure 7**), dig out some information from it, and try to find the cause of this crash.

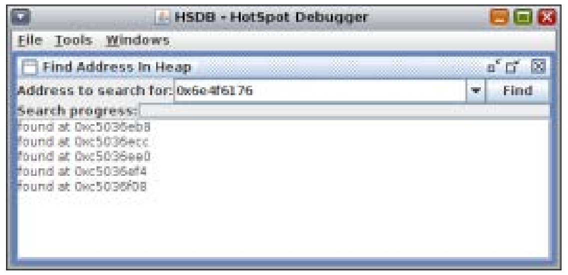
Figure 8 shows the disassembly of the code that was being executed around PC 0xfe5d2c17 when the crash happened.

The instructions shown in **Figure 8** indicate that the process crashed when trying to access the value at address eax+100. From the hs_err file, we can



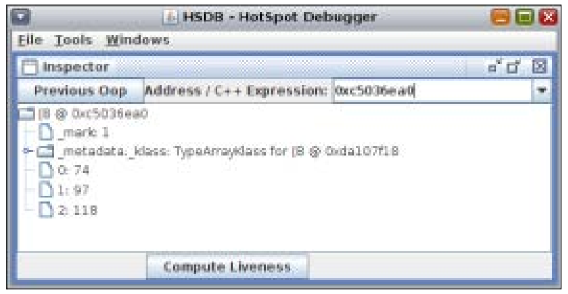
```
HSDB - HotSpot Debugger
File  Tools  Windows
Find Pointer
Address: 0x6e4f6176
Address 0x6e4f6176: In unknown location
```

Figure 9



```
HSDB - HotSpot Debugger
File  Tools  Windows
Find Address In Heap
Address to search for: 0x6e4f6176   Find
Search progress:
found at 0xc5036eb8
found at 0xc5036ecc
found at 0xc5036ea0
found at 0xc5036ef4
found at 0xc5036f08
```

Figure 10



```
HSDB - HotSpot Debugger
File  Tools  Windows
Inspector
Previous Oop   Address / C++ Expression: 0xc5036ea0
[B @ 0xc5036ea0
  _mark: 1
  _metadata._klass: TypeArrayKlass for [B @ 0xda107f18
  0: 74
  1: 97
  2: 118
Compute Liveness
```

Figure 11

see the contents of the registers and what the value of the EAX register was:

EAX=0x6e4f6176, EBX=0xc50a083c,
ECX=0x614a2e2e, EDX=0x00000006
ESP=0xfbc7e360, EBP=0xfbc7e398,
ESI=0xc5036ef0, EDI=0x00000000
EIP=0xfe5d2c17, EFLAGS=0x00010202

What was at 0x6e4f6176, and why did the crash happen while reading the value at this address? HSDB helps us see that, as shown in **Figure 9**.

The address does not lie in the Java heap. Using the **Find Address in Heap** option, we can find the locations in the Java heap from which this particular

LISTING 7 / LISTING 8 / LISTING 9 / LISTING 10 / LISTING 11 / LISTING 12

```
java -Dsun.jvm.hotspot.debugger.useWindbgDebugger=true -classpath d:\java\
jdk1.7.0_03\lib\sa-jdi.jar sun.jvm.hotspot.tools.PermStat 5684
Attaching to process ID 5684, please wait...
Debugger attached successfully.
Client compiler detected.
JVM version is 22.1-b02
10713 intern Strings occupying 802608 bytes.
finding class loader instances ..
done.
computing per loader stat ..done.
please wait.. computing liveness..........................................done.
class_loader   classes bytes   parent_loader   alive?  type

<bootstrap>    342    1539808  null        live    <internal>
0x23f7b398     3     28016   0x23f762e0     live    sun/misc/Launcher$AppClassLoader
@0x38a0e9c0
0x23f762e0     0     0       null        live    sun/misc/Launcher$ExtClassLoader@0x389
eb420

total = 3      345    1567824     N/A       alive=3, dead=0     N/A
```

Download all listings in this issue as text

address is referenced (see **Figure 10**).

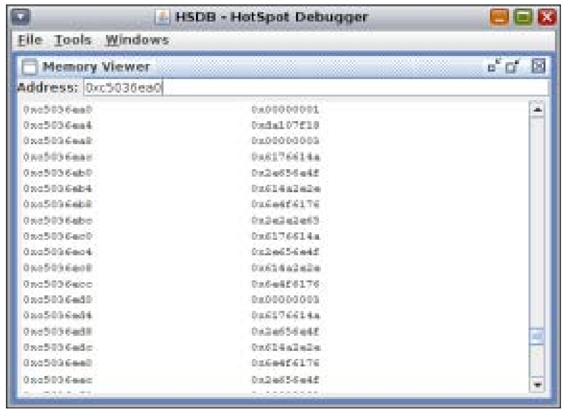
Examine these found locations in the Object Inspector to see if these are part of any object, as shown in **Figure 11**.

All the found addresses bring up the byte array object at 0xc5036ea0 in the Object Inspector, which means the object at 0xc5036ea0 is the closest valid

LISTING 13 LISTING 14

```
(dbx) x 0xc5036ea0/100c
0xc5036ea0:  '\001' '\0' '\0' '\0' '\030' '\0177' '\020' '□' '\003' '\0' '\0' '\0'
'H' 'e' 'l' 'l'
0xc5036eb0:  'o' ' ' 'J' 'a' 'v' 'a' '.' 'H' 'e' 'l' 'l' 'o' ' ' 'J' 'a' 'v'
0xc5036ec0:  'a' '.' 'H' 'e' 'l' 'l' 'o' ' ' 'J' 'a' 'v' 'a' '.' 'H' 'e' 'l'
0xc5036ed0:  '\003' '\0' '\0' '\0' 'a' 'v' 'a' '.' 'H' 'e' 'l' 'l' 'o' ' ' 'J' 'a'
0xc5036ee0:  'v' 'a' '.' 'H' 'e' 'l' 'l' 'o' ' ' 'J' 'a' 'v' 'a' '.' 'H' 'e'
0xc5036ef0:  'l' 'l' 'o' ' ' 'J' 'a' 'v' 'a' '.' 'H' 'e' 'l' 'l' 'o' ' ' 'J'
0xc5036f00:  'a' 'v' 'a' '.'
```

Download all listings in this issue as text



```
0xc5036ea0          0x00000001
0xc5036ea4          0xda107f18
0xc5036ea8          0x00000003
0xc5036eac          0x6176614a
0xc5036eb0          0x2e656c4f
0xc5036eb4          0x614a2e2e
0xc5036eb8          0x6e6f6176
0xc5036ebc          0x2e2e2e65
0xc5036ec0          0x6176614a
0xc5036ec4          0x2e656c4f
0xc5036ec8          0x614a2e2e
0xc5036ecc          0x6e6f6176
0xc5036ed0          0x00000003
0xc5036ed4          0x6176614a
0xc5036ed8          0x2e656c4f
0xc5036edc          0x614a2e2e
0xc5036ee0          0x6e6f6176
0xc5036ee4          0x2e656c4f
```

Figure 12

object just before these locations. If we look carefully, these locations actually go beyond the limits of the byte array object, which should end at 0xc5036eb0, and from address 0xc5036eb0, the next object should have started. See the raw contents at memory location 0xc5036ea0 in **Figure 12**.

We can look at the raw contents as characters in the dbx debugger. See **Listing 13**, which clearly shows that the object at 0xc5036ea0 has a byte stream that goes beyond its size limit of three elements and overwrites the object starting at 0xc5036eb0.

This gives us a big clue. Now, we can easily search in the code where the bytes "Hello Java.Hello Java. . ." are being written, and find the buggy part of the code that overflows a byte array. **Listing 14** shows the faulty lines that I had in my JNI code. Wow! This was so easy.

Summary

As in the example above, we in the JVM Sustaining Engineering Group at Oracle use the Serviceability Agent on a daily basis to debug crashes, hangs, and other kinds of problems that occur with the Java HotSpot VM. SA is a pretty useful and powerful debugging tool that can also help you learn the internals of the Java HotSpot VM. I hope this article provided good insight into this tool. Enjoy debugging with SA! `</article>`

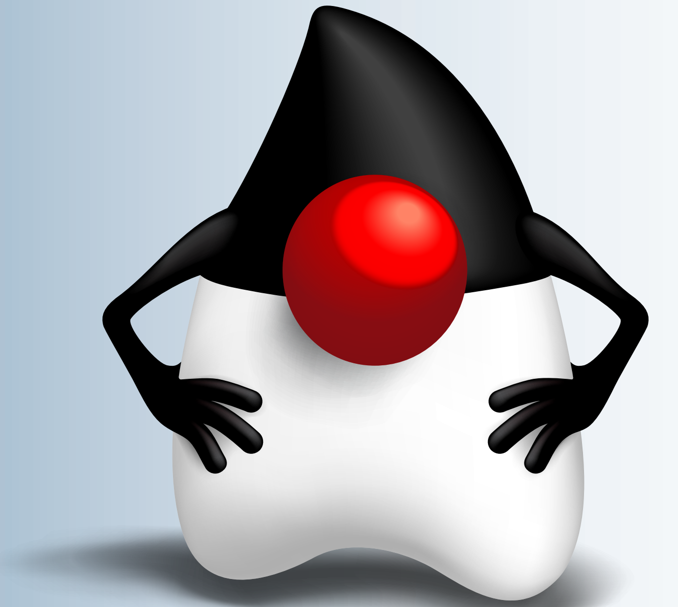
LEARN MORE

- SA-Plugin for VisualVM

Java™

YOUR LOCAL JAVA USER GROUP NEEDS YOU

Find your JUG here



JOSH MARINACCI

BIO

Fork/Join Framework for Client Java Applications

Java SE 7's fork/join framework makes a great match for CPU-intensive client-side applications.

CPUs aren't getting faster. Nearly ten years ago, we hit 3 GHz and I have yet to own a 4 GHz machine, much less the 43 GHz Moore's Law promised me. CPUs can do far more than they used to, but they do it by getting wider instead of faster—more CPUs with more cores with more threads.

While this is great for the end user, it introduces new challenges for application developers. I can buy a 12-core desktop from Apple, and even my small laptop has 2 cores now. The world is becoming parallel, so we need new ways to code for parallel machines.

Java has always had good support for concurrent programming thanks to java.lang.Thread. The Thread class has existed since the first release of J2SE 5.0 introduced the ExecutorService to ease the management of groups of threads. Unfortunately, threads mainly help with I/O-bound tasks. That's fine for server-side jobs, such as hosting Web apps, but modern desktop apps need help with CPU-bound tasks as well.

It is far harder to make CPU-bound tasks maximize your computer, especially on a general-purpose operating system where many other processes vie for CPU time. To help with this, Java SE 7 introduced a new framework to the concurrency utilities that can help with CPU-bound tasks: the fork/join framework.

The fork/join framework in Java SE 7 is conceptually pretty simple. You fork the current thread to divide up your work, and then you join the tasks back together to collect the finished results. As we will see below, coding with this new framework is pretty easy. The value of fork/join is what it does under the hood.

Fork/Join Framework

The fork/join framework provides three benefits over the Executor interface.

- Fork/join is a natural fit for a recursive algorithm, especially if you don't know the scope of the work beforehand.
- Fork/join provides work-stealing behavior that can better balance the workload across multiple processors, and with less lock contention. Traditional algorithms can max out at about eight processors before the overhead of lock contention outweighs the speed of using more cores. Fork/join can scale to over 100 cores.
- Fork/join is future-proof and portable. The code you write with it doesn't assume anything about the underlying hardware. It's designed to run your code as efficiently as possible across any hardware—both today's dual-core laptops and single-core phones, as well as the 100-core desktop of the future.

ALGORITHM AT WORK

Every class that uses the fork/join framework works with pretty much this algorithm: If the amount of work is below a threshold, then do the work; otherwise, split the work in half and recurse, waiting for each half to complete.

Something Simple

To see how the fork/join framework works, let's try something simple. Every class that uses the fork/join framework works with pretty much this algorithm (see **Listing 1**): If the amount of work is below a threshold, then do the work; otherwise, split the work in half and recurse, waiting for each half to complete. This is the textbook divide-and-conquer method.

To demonstrate how this works in practice, I've created a simple example that calculates the minimum number from a very large array of doubles (see **Listing 2**).

PHOTOGRAPH BY CHRIS PIETSCH/GETTY IMAGES

The compute method in **Listing 2** implements the algorithm I described in **Listing 1**. If the number of doubles to be searched is less than a threshold (100), then calculate the minimum; otherwise, recurse on each half and wait for them to join (complete). Once you have the result of each half, calculate the minimum of those and return it.

The MinimumFinder class implements the RecursiveTask<Double> interface, which is one of two interfaces defined by the fork/join framework. The other interface, RecursiveAction, is identical except that its compute method doesn't return anything.

The magic is in the join() method. It will wait for the subtasks to complete. **Figure 1** shows a simple benchmark of the time it takes to search through 30 million random numbers using a single core or the two cores on my laptop. I calculated the average over 10 runs for a single core, and then switched to doing it again with two cores and repeated that six times. By duplicating and averaging, I can ensure that I get reliable results.

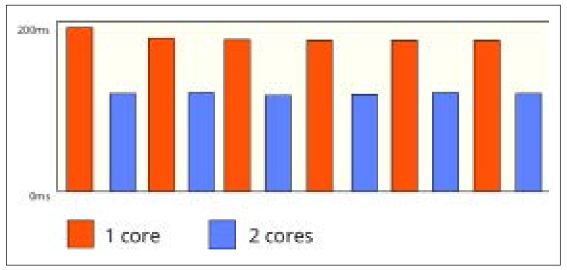


Figure 1

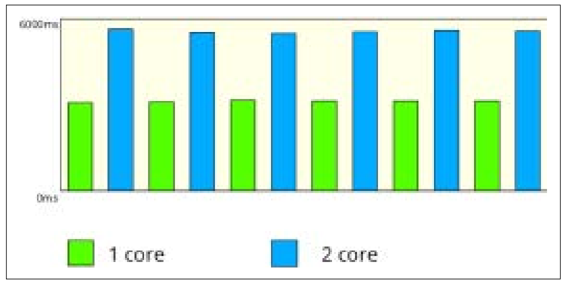


Figure 2

You can see in **Figure 1** that the single-core time hovers around 190 ms and the dual-core time hovers around 122 ms, which means that two cores take only 64 percent of the single-core time. The improvement is not quite the double that we might hope for when using two cores, but it's pretty good.

Why weren't the dual-core results exactly twice as fast (95 ms)? There are a few reasons. First, there is overhead involved in using the fork/join framework. For small tasks, it's simply not worth the overhead. Unfortunately, most tasks for which it's worth the overhead are also too complicated to use as examples, which is why many of the fork/join tutorials on the Web demonstrate something like calculating the Fibonacci series. Calculating the Fibonacci sequence with recursion is actually a horrible example because it's not very efficient—you won't get a 2x speedup, and there are much better nonrecursive ways to do it faster.

That said, how could we improve this demo to better show the value of multicore? Let's make the algorithm do more work so that the fork/join overhead is less of a factor. I kept the same code but added a line that does three multiplications per value in the array. Because multiplication is much slower than comparisons, this should increase the calculation time a lot. **Figure 2** shows the results of the next run, now slow enough to take nine minutes on my laptop for the entire run.

LISTING 1 LISTING 2

```
compute() {
  if ( work.size < threshold ) {
    return doWork(work);
  } else {
    f1 = fork(first half of work)
    f2 = fork(second half of work)
    wait for the forks to join
  }
}
```

Download all listings in this issue as text

The dual-core version takes about 55 percent of the time that the single-core version takes. That's within about 10 percent of perfect efficiency, and it constantly pegs my CPU. Excellent!

This example is fine for learning the API, but what could we actually do with this in the real world—especially when it comes to client-side applications? The first thing that comes to mind is graphics. A lot of complex graphics rendering is very easy to make parallel. Let's take a look at one of the most common CPU-intensive kinds of graphics: fractals.

Parallel Mandelbrot Graphics

The Mandelbrot set is often considered *the* classic fractal or at least the most widely known. Every pixel in the Mandelbrot set can be calculated independently of the others, which makes it a great target for parallel computation.

It also has another interesting attribute: the work isn't smooth. To calculate a pixel in the fractal, you run a simple equation over and over again until the value exceeds a certain threshold. This is called *escaping*. The color of the pixel is based on how many times through the loop it took for the value to escape. If the loop goes on long enough (another threshold value), then we can say that the pixel never escapes and mark it as black. **Figure 3** shows a classic Mandelbrot set fractal.

Take a look at this picture of the Mandelbrot set. The center part is filled with pixels that never escape. This means they each went through the loop the same number of times (to the threshold), so the work required to calculate those pixels is always the same.

The colored pixels, however, escaped sometime before the limit. This means that these pixels required less work than the maximum amount. Of course, the black pixels are boring. It's the varying colored pixels that are interesting, so the part we want to look at has the most varied workload. The fork/join framework is the perfect way to parallelize this workload efficiently.

A regular parallel algorithm for the Mandelbrot set would divide the picture into rows of pixels, allocating a thread for every set of rows. So what would happen if one set of rows happened to take longer than the other set? This will happen if one row in the fractal has more black in it than another.

PERFECT FIT

Fork/join is a natural fit for a recursive algorithm, especially if you don't know the scope of the work beforehand.

If we used a regular thread pool, one of the threads would be done before the others and would just sit around being idle. A fork/join thread pool wouldn't, however, because of a very unique property: *work stealing*. If one of the threads is idle because it has finished its work, it can steal some work from another thread. The framework will automatically load balance across the set of threads, making sure it's always crunching on something and maximizing the CPUs.

Let's look at the Mandelbrot set using

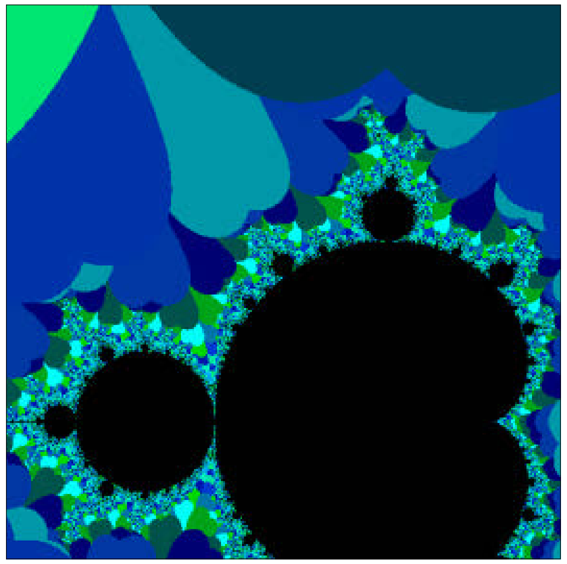


Figure 3

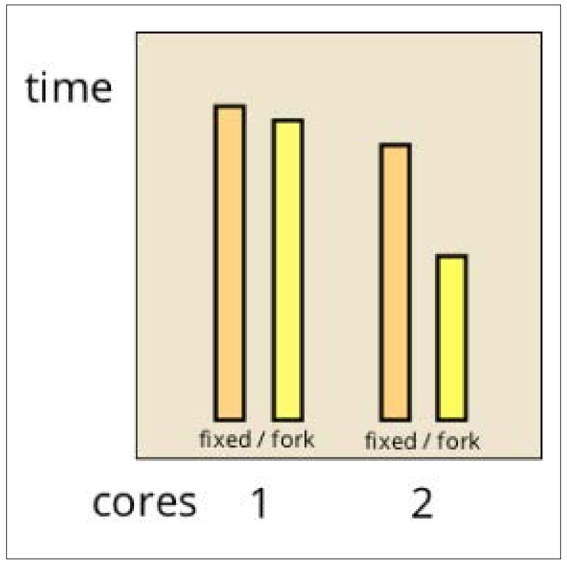


Figure 4

a fixed thread pool versus the fork/join framework. **Figure 4** is a graph of the results of the fixed thread pool versus the fork/join framework for a 4,000 x 4,000 pixel fractal.

We can loosely group workloads into two categories: smooth and lumpy. With a smooth workload, all units of work take about the same effort (time) to complete, whereas lumpy workloads take variable—often highly variable—amounts of effort. You can see this effect in **Figure 4**. While both versions are faster when using two cores, the fork/join version finishes faster than the pool version. The pool version is only a tiny bit faster with two cores than with one core because the work is lumpy. On a two-core machine, the first thread finishes three times faster than the second thread, but because it is a fixed pool, you are limited by the time of the slowest thread. The fork/join version doesn't have this problem with lumpy workloads thanks to work stealing.

Oracle's concurrency expert, Brian Goetz, explains it this way:

"If you've got a problem of size N with p processors, when you divide into problems of size N/p you are sure to get lumpiness. Some will finish first and those CPUs will then be idle. If you make subproblems of, say, N/10*p, the single work queue becomes a sequential bottleneck as multiple threads contend for 'get next chunk of work.' Even if your problem is evenly balanced to start, things like cache misses, page faults, GC, etc., will cause different ones to run at different rates. You get lumpiness. Fork/join says: rather than sweating the lumpiness, let work stealing iron it out for you."

A quick note on this example. I am using a buffered image to store the pixels. Each loop would call image.setRGB for each result pixel. Commenting out the image.setRGB line speeds up my test of even the single-core version from an average of 5,000 ms to 3,900 ms.

Something in the way buffered images work is very slow compared to just setting an array value. This shows that the things we think are slow aren't always the culprit. In this case, I could improve the general code performance a lot by using an array of integers and then converting to an image after the work is done. The lesson here is that we must always profile our code to figure out where it is really slow.

IMAGE HANDLER

The new fork/join framework is a great match for **CPU-intensive client-side applications**, such as image manipulation and image generation.

Image Segmentation

For a final example, I chose something a little bit different: image segmentation. Many algorithms in image processing, from compression to computer vision, begin with the step of dividing the image into chunks that are similar. This is called *image segmentation*. Once an image is segmented, you can perform lots of additional processing to find human faces, look for object edges, and do many other things; so it's good to have fast image segmentation.

One simple form of image segmentation uses a *quadtree graph*. To segment the image, you start by looking at a large square section of the image (usually the entire image) and determine if the pixels in that square are similar enough using a threshold value. If so, then the square can be represented with a single color value. If they are not similar enough (which is most likely, initially), then subdivide the square into four smaller squares and repeat. Eventually, this recursive algorithm will reach small enough squares so that the pixels within are similar, or you reach a square the size of a single pixel (which of course is similar to itself). The resulting graph of nested squares is called a *quadtree* (the three-dimensional version of it is called an *octree*).

Figure 5 shows an image with the quadtree graph overlaid on top. You can see that there are more small squares in the parts with more detail.

Some of the squares in the quadtree will be larger than others, because that part of the image had more similar pixels. The parts of the image with more different pixels will have smaller squares and will be deeper in the graph. This means the graph construction

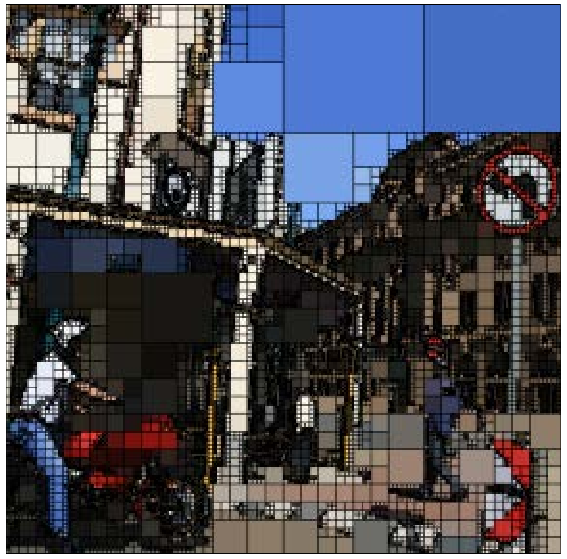


Figure 5

YOUR LOCAL JAVA USER GROUP NEEDS YOU

Find your JUG here

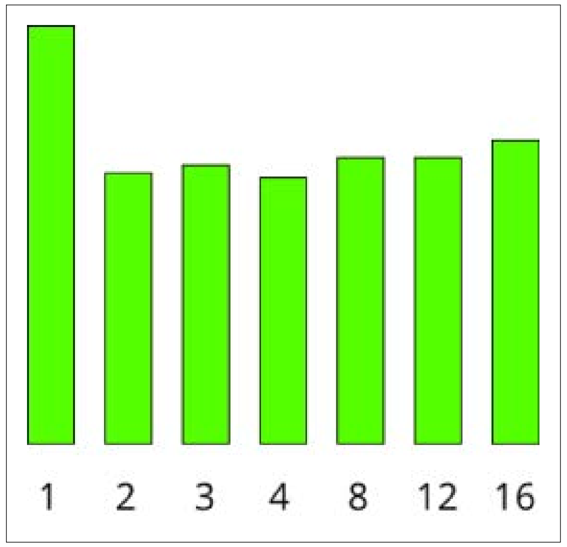


Figure 6

process will be lumpy. Some parts of the image will take longer to process than others, using more CPU. Most importantly, we don't know beforehand which parts of the image will take longer, so we can't evenly divide the work among CPUs without the work-stealing behavior of the fork/join framework. So, image segmentation boils down to recursive graph construction comparing lumpy image data to a threshold value. Sounds perfect for fork/join.

Take a look at **Listing 3** to see a demo app and performance tests. I adapted this example from the code presented in this Dev.Mag article. I don't include the code for the entire project; I provide just the interesting parts. As a side note, it only took about five minutes to parallelize the original code. The fork/join framework plays very nicely with recursive algorithms.

I ran this app on my dual-core laptop (with a larger 2,000 x 3,000 test image) set to run with different amounts of threading. See **Figure 6**, which shows the runtimes with different thread counts.

You can see that the speedup between a single thread and two threads is quite significant. It doesn't speed up after that, of course, because I have only two physical cores. However, it doesn't slow down much either. This shows that the fork/join framework can do its job with minimal overhead.

Conclusion

The fork/join framework doesn't actually make the Java Virtual Machine run faster. It simply presents a more convenient way to express parallel algorithms so that your code can run more efficiently than if you had to split it up by hand. The real point of fork/join is that you have to worry only about your algorithm, not how it's implemented. Your code could run on one core or a hundred, and you don't have to care. The system will maximize efficiency for you, now and in the future. This makes the new fork/join framework a great match for CPU-intensive client-side applications, such as image manipulation and image generation. </article>

LEARN MORE

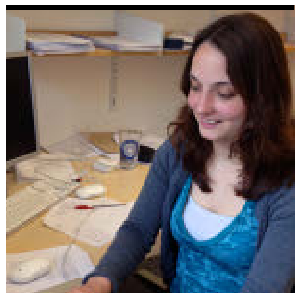
- Josh on Design
- Fork/join framework
- "Showtime! Java 7 Is Here!"
- "Meet JavaMan"
- "Fork and Join: Java Can Excel at Painless Parallel Programming Too!"

LISTING 3

The following listing has been excerpted for space, as noted by the . . . symbol. The full code listing is available by downloading the code listings for this issue.

```
public class Node {
...
  protected void compute() {
    if(w <= 1 || h <= 1 || measureDetail(grid,x,y,w,h) <= threshold) {
      color = average(grid,x,y,width,height);
    } else {
      children[0] = new Node(grid,x,y,w/2,h/2,threshold);
      children[1] = new Node(grid,x+w/2,y,w-w/2,h/2,threshold);
      children[2] = new Node(grid,x,y+h/2,w/2,h-h/2,threshold);
      children[3] = new Node(grid,x+w/2,y+h/2,w-w/2,h-h/2,threshold);
      invokeAll(children);
    }
  }
  Color average(int[][] grid, int x, int y, int w, int h) {
    int redSum = 0; int greenSum = 0; int blueSum = 0;
    int area = w*h;
    for(int i=x; i<x+w; i++) {
      for(int j=y; j<y+h; j++) {
        redSum += getRed(grid[i][j]);
        greenSum += getGreen(grid[i][j]);
        blueSum += getBlue(grid[i][j]);
      }
    }
    return new Color(redSum/area,greenSum/area,blueSum/area);
  }
  int measureDetail(int[][] grid, int x, int y, int w, int h) {
    Color avg = average(grid, x, y, w, h);
    int red = avg.getRed(); int green = avg.getGreen(); int blue = avg.getBlue();
    int area = w * h; int colorSum = 0;
    for(int i=x; i<x+w; i++) {
      for(int j=y; j<y+h; j++) {
        colorSum += Math.abs(red-getRed(grid[i][j]));
        colorSum += Math.abs(green-getGreen(grid[i][j]));
        colorSum += Math.abs(blue-getBlue(grid[i][j]));
      }
    }
    return colorSum / (3 * area);
  }
```

Download all listings in this issue as text



RAOUL-GABRIEL URMA, **JANINA** VOIGT, AND **MARTIJN** VERBURG

BIO

MARTIJN VERBURG'S PHOTOGRAPH BY BOB ADLER

How to Modify javac

Learn the steps for changing javac to implement new language syntax features.

For Java developers who want to work with the language and platform they love, the last two years have been groundbreaking ones. Java SE 7 was released after a long five-year wait and—crucially for developers interested in Java and the Java Virtual Machine (JVM) itself—the OpenJDK project became the Reference Implementation (RI) for Java.

In addition to the RI code base becoming fully open source, great progress was made in simplifying the complex build process, and most developers are now able to build the OpenJDK with a simple one-line command.

Last, the existing OpenJDK committers continued their outreach to the wider Java community, encouraging participation. Today, the OpenJDK is more open than ever and the barriers to entry have been significantly lowered.

In late April 2012, the London Java Community launched the Adopt OpenJDK program. This program is a group of projects and events led by Java user groups (JUGs) that aims to channel into the OpenJDK the efforts of the wider Java community (those members of the Java community that are not already directly involved in the OpenJDK, such as Oracle, Red Hat, IBM, and many others), while keeping in mind a "do no harm" policy.

Flooding the OpenJDK project with thousands of patches a week and demanding acceptance of ill-thought-out language features would not make for a better Java. For the Adopt OpenJDK program, doing no harm means producing well-thought-out, peer-reviewed bug fixes and new features that are backed by empirical evidence and coordinated with the core OpenJDK committers.

Why such a formal/strict approach? The OpenJDK is now the heart of a vital piece of technology that (in application software terms) runs large parts of our entire civilization, affecting billions of people daily. So, changes to it need to be made with a great deal of care and with as much scientific rigor and empirical analysis as possible.

That's not to say that the day-to-day developer can't get involved, and this article takes you through implementing a relatively simple new language syntax feature, the *Elvis operator*. It highlights *some* of the technical steps that you would take if you were to tackle one of the more advanced projects in the Adopt OpenJDK program. Of course, to add a language feature, you also would have to fulfill the requirements of the OpenJDK process by submitting a JDK Enhancement Proposal (JEP).

Overview of Compilation Using javac

The Java compiler, javac, takes a set of Java source files (.java) as input and produces the corresponding .class files as output. You need to understand the basics of javac compilation in order to add features to Java. This process is performed in three distinct phases, as shown in **Figure 1**. We'll just give you a quick overview of the three compilation phases here; for more details, see the OpenJDK documentation.

Parse and Enter phase. As a first step, the javac lexer reads the Java source files (as an input stream of characters) and maps these into a sequence of tokens. The parser takes these tokens and generates Abstract Syntax Trees (ASTs) representing the source program. These trees are made of AST nodes that represent the different constructs in the source code, such as method declarations, statements, and so on.

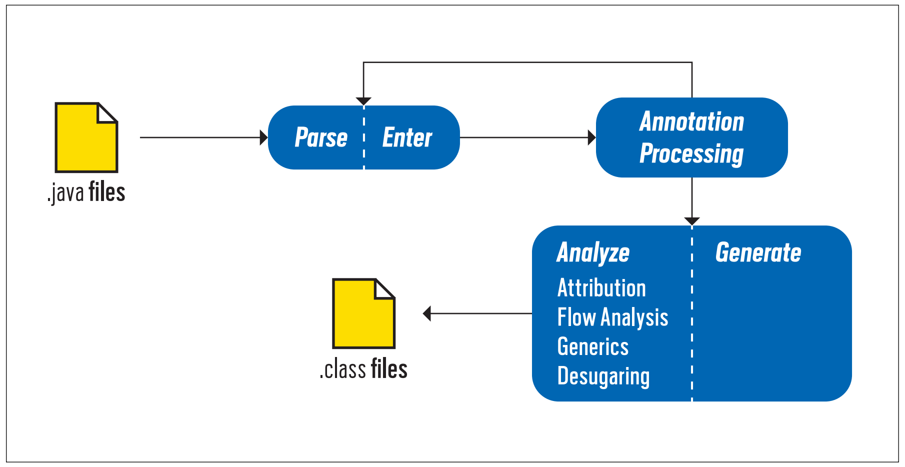


Figure 1

For example, consider the simple expression 2 + 2 * 5. We can represent this expression as the AST shown in **Figure 2**. The tree structure makes it clear how (and in what order) the expression is evaluated.

Once the parser has generated ASTs, any symbols defined by the program are entered into the symbol table. For each class, all symbols defined by the class are entered into the class' scope.

Annotation Processing phase. In the second phase, any annotations in the program are processed by calling the relevant annotation processors. Each annotation processor is loaded and run in a separate class loader. It is possible that running an annotation processor generates additional source files; if this happens, the Parse and Enter phase might need to be repeated.

Analyze and Generate phase. Next, javac analyzes the ASTs before generating .class files. The analysis of the ASTs entails the following steps:

1. Attribution of ASTs: Names and expressions in the ASTs are resolved, type checking is performed, and constant expressions are simplified (a process known as *constant folding*). Many semantic errors can be discovered during this phase, including type errors and missing symbols.
2. Flow analysis: The javac compiler checks that all statements are reachable and performs exception analysis to ensure that every checked exception that is thrown is either declared or caught. It also analyzes assignments to ensure that each variable is assigned a value before it is used and that final variables are assigned values only once.
3. Generics: Code that includes generics is translated into code without generics to comply with type erasure.
4. Syntactic desugaring: The javac compiler rewrites the syntax trees to remove syntactic sugar, such as inner classes, assertions, and for-each loops.

If all the checks above pass, javac generates .class files containing Java bytecode.

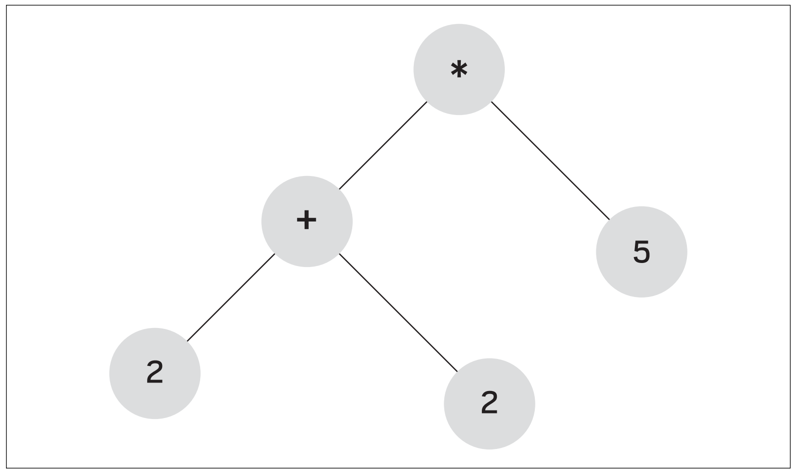


Figure 2

Adding the Elvis Operator to Java

In this section, we'll show you, step by step, how to modify javac to implement the Elvis operator.

The Elvis operator, ?: (look at it sideways to see where it got its name), is useful for returning a default value when a variable is null. As explained in "Elvis and Other Null-Safe Operators," it is a binary operator and "results in the value of the left-hand side if it is not null, avoiding evaluation of the right-hand side. If the left-hand side is null, the right-hand side is evaluated and is the result."

For example, suppose we have a String foo:

```
foo ?: "default"
```

It returns the String value stored in foo if foo is not null; otherwise, it returns default.

Using the existing ternary operator (which has a very similar syntax), the previous statement is equivalent to this:

```
(foo != null) ? foo : "default"
```

The Elvis operator is available in several other JVM languages, including Groovy and Scala, but not in Java itself. It was proposed as an addition to JDK 7 as part of Project Coin but was not included in the final feature selection. As we noted earlier, adding even seemingly simple language features needs to be weighed carefully.

Building the Java Compiler

Building javac is fairly simple. Visit the OpenJDK compiler group for an overview. We've distilled the steps below:

1. If it is not already installed, install a Mercurial DVCS client for your operating system.
2. If it is not already installed, install the ANT build tool.
3. Go to the command line (or your Mercurial GUI client).
4. Enter the following command:

```
hg clone http://hg.openjdk.java.
net/jdk8/tl/langtools <dir>
```

Once the code has been cloned into the *<dir>* directory, do the following:

1. Enter the following command:

```
cd <dir>/make
```

2. Edit the build.properties file and set boot.java.home to an installation of the JDK you already have.
3. Enter the following command:

```
ant
```

The build should complete with a message similar to the following:

```
BUILD SUCCESSFUL
Total time: 1 minute 2 seconds
```

Modifying the Java Compiler

The implementation of the Elvis operator in the Java language requires several steps:

1. Modify the lexer so that it recognizes the Elvis operator.
2. Modify the parser so that it generates an AST node representing the Elvis operator.
3. Provide a translation of the Elvis operator in Java bytecode.

However, this process can be simplified slightly because the Elvis operator can be converted to an existing language feature: the ternary operator. Therefore, we can just modify the parser so it generates a ternary operator AST node. We can then rely on the translation phase to recognize this node and translate it to correct Java bytecode without providing our own translation implementation (this process is called *AST translation*). Let's begin with the first step, the lexer modification.

Lexer modification. First, let's add the Elvis operator so it is recognized as a new token. We do this by modifying the com.sun.tools.javac.parser.Token class, inserting the Elvis operator as a new token value. See **Listing 1**.

All the tokens are automatically added to a keyword table, which is defined in the com.sun.tools.javac.parser.Keywords class. The Keywords class maps all token names (the value passed into the Token constructor) to their Token instance.

The keyword table is then used by the Java lexer to map an input stream of characters into a token sequence. The lexer interface is specified in com.sun.tools.javac.parser.Lexer and it contains various methods, such as the following:

- pos(), which returns the current position of the lexer
- nextToken(), which reads the next token
- token(), which returns the current token, typically using the default implementation that lies in com.sun.tools.javac.parser.Scanner

Parser modification. We now modify the Java parser so that it recognizes expressions of the form Expression1 ?: Expression2. The implementation for this is defined in com.sun.tools.javac.parser.JavacParser.

There's a method in JavacParser called parseExpression(), which parses expressions. It processes the token stream it gets from the Java lexer and returns AST nodes as output.

BIG IMPACT

The OpenJDK is now the heart of a vital piece of technology that (in application software terms) runs large parts of our entire civilization, affecting billions of people daily.

Each AST node is a subclass of com.sun.tools.javac.tree.JCTree. Since parseExpression() deals only with expressions, it outputs a JCExpression (a subtype of JCTree) that represents expression nodes. Other kinds of nodes are represented by other subclasses of JCTree. For example, statement nodes are represented by JCStatement.

The TreeMaker class is a factory for AST nodes; JavacParser has a field called F of type TreeMaker that we can use in our code to easily create various kinds of AST nodes.

The parseExpression() method is defined as shown in **Listing 2** and essentially sets the parser mode to expect expressions (EXPR). It then calls term() to parse and create expression nodes of type JCExpression.

The term() method distinguishes between assignment expressions and other types of expressions. For example, x = 1 and x /= 5 are assignment expressions; x++ counts as another expression.

If term() discovers an assignment operator in the expression, it parses the left-hand side and the right-hand side of the assignment separately by first calling term1() and then termRest(); of course, both of the sides can be made of subexpressions. For all other kinds of expressions, all the parsing is delegated to term1(), which we'll look at more closely now.

The term1() method is defined as shown in **Listing 3**.

We can see that term1() delegates most of its work to yet another helper

LISTING 1 LISTING 2 LISTING 3

```
public enum Token implements Formattable {
    EOF,
    ERROR,
...
    QUES("?"),
    ELVIS("?:"),    // <---
    COLON(":"),
...
}
com.sun.tools.javac.parser.Token
```

Download all listings in this issue as text

method, term2(). Fortunately for us, we won't need to look more closely at term2() here, because term1() handles the parsing of the ternary operator. If term1() finds a question mark token (QUES), it knows it's dealing with a ternary operator. The first part (before the question mark) has already been parsed by term2(); the remaining two parts (before and after the colon) are then parsed by term1Rest().

The Elvis operator is very similar to the ternary operator, so term1() is a good place to parse it. So if we find an Elvis token here, we parse the expression as an Elvis operation, as shown in **Listing 4**.

Our task now boils down to providing the implementation for term1Elvis(JCExpression). As explained previously, we can simply convert an Elvis operation into a ternary operation. Thus, term1Elvis(JCExpression) should parse the right-hand side of the expression (which follows the ?: token) and then create an AST node representing a ternary operation of the form expr1 != null ? expr1 : expr2.

We already have an AST node for the left-hand side of the Elvis operation (passed as argument expr1). We then parse the right-hand side by calling term() and store this in expr2.

The next step is to construct an AST node representing the conditional expression expr1 != null. Now we have the three pieces that are required, so we can just create the ternary operation AST node. This is done by calling the Conditional method of the TreeMaker and passing in the conditional node and the nodes representing the left-hand side and right-hand side of the expression.

The final implementation of term1Elvis() is shown in **Listing 5**.

Now that you've finished the implementation, rebuild the javac compiler, per the "Building the Java Compiler" section. Congratulations; you can now use the Elvis operator in your code with your shiny new Java compiler!

Testing

Of course, as with any software, we need to ensure that there's some level of testing. (Some would argue we could have used test-driven development here, but we were influenced by Martijn, the Diabolical Developer.)

So, let's write a basic test to see if the Elvis operator works as desired. See **Listing 6**.

The code in **Listing 6** should print the word *unknown*.

Note that because we simply translated our Elvis operator into a ternary operator, we don't need to worry about things like type checking. For example, the code in **Listing 7** doesn't compile (as expected). The error shown in **Listing 8** is reported.

Conclusion

Adding the Elvis operator is a fun example of adding a small language feature at the level of the javac compiler. If this was a serious attempt at getting this feature into the OpenJDK, you would have to apply more-rigorous proof that the code was correct and safe as well as providing empirical evidence (such as using the Qualitas Corpus) that this feature would be utilized by developers. The participants in the Adopt OpenJDK program are there to help with all of that.

We hope you enjoyed this journey into javac, and we hope to see you in the Adopt OpenJDK program working on the language and the platform that we all share. </article>

Raoul-Gabriel Urma and Janina Voigt are PhD students from the department of Computer Science at Cambridge University, U.K. They have embarked on several research topics related to Java and have joined the Adopt OpenJDK program. They have very kindly made some of their research and methodology available to share with fellow developers. It's been a pleasure to collaborate on this article with them; I certainly learned a great deal. —Martijn Verburg

LISTING 4 / LISTING 5 / LISTING 6 / LISTING 7 / LISTING 8

```
JCExpression term1() {
  JCExpression t = term2();
  if ((mode & EXPR) != 0 && S.token() == QUES) {
    mode = EXPR;
    return term1Rest(t);
  }
  else if ((mode & EXPR) != 0 && S.token() == ELVIS)
  {
    mode = EXPR;
    return term1Elvis(t);
  }
  else {
    return t;
  }
}
```

Download all listings in this issue as text

JAMES L. WEAVER

BIO

PHOTOGRAPH BY STEVE GRUBMAN

Part 2

Lazy Evaluation, Lazy Initiation, and Custom Bindings in JavaFX 2

Optimize the evaluation of bindings and the initialization of properties the lazy way.

JavaFX 2 is an API and runtime for creating rich internet applications (RIAs). It was introduced in 2007, and version 2 was released in October 2011. One of the advantages of JavaFX 2 is that the code can be written in the Java language, using mature and familiar tools.

This article is Part 2 of a two-part series and focuses on optimizing JavaFX 2 properties and binding by implementing lazy evaluation, lazy initialization, and custom bindings.

As discussed in the previous article, "Using Properties and Binding in JavaFX 2.0: Part 1," JavaFX 2 comes with a set of interfaces, which are shown in **Figure 1**. The purpose of these interfaces is to provide support for using and implementing properties, detecting when the values of properties have changed, and binding properties to other properties.

These interfaces are located in four packages:

- javafx.beans
- javafx.beans.binding
- javafx.beans.property
- javafx.beans.value

This article contains an example of using the methods defined by many of these interfaces to implement lazy evaluation, lazy initialization, and custom bindings.

Overview of the LazyInitEvalSolution Application

To help you learn how to use properties and binding, an example application named LazyInitEvalSolution will be employed. As shown in **Figure 2**, this application contains three stopwatches, and each has some buttons and elapsed and lap time displays.

The LazyInitEvalExercise project that you'll download in the next section contains starter code for the example application. In its current form, the application's runtime appearance is similar to **Figure 2**. During the course of this article, you'll modify the code to implement the lazy evaluation, lazy initialization, and custom binding behavior of the LazyInitEvalSolution project, which is also available in the file you'll download.

As shown in **Figure 3**, when you click the **Start** button on one of the stopwatches, its elapsed timer starts counting by milliseconds. In order to see the elapsed time, however, you must click the invalidate – click to bind/unbind

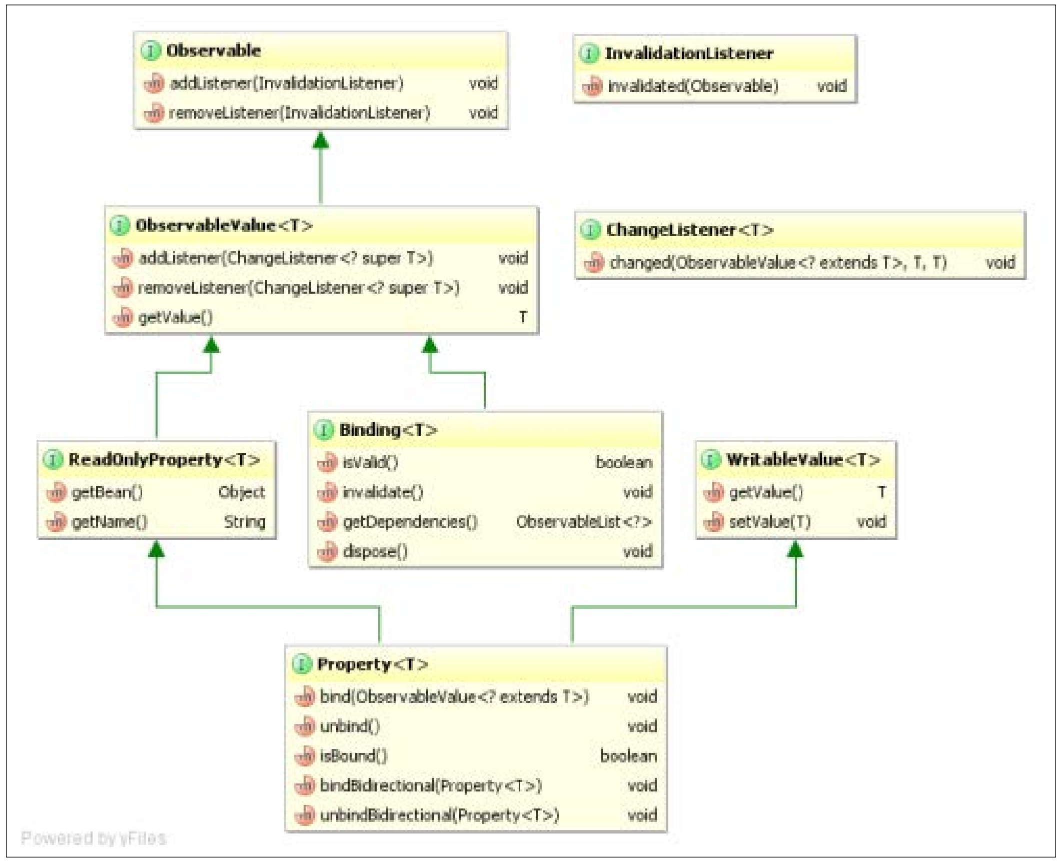
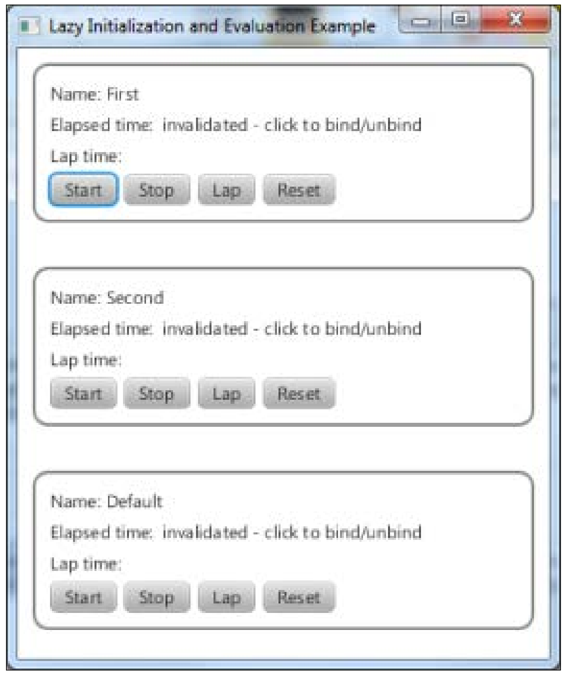

Observable
addListener(InvalidationListener) void
removeListener(InvalidationListener) void
InvalidationListener
invalidated(Observable) void
ObservableValue<T>
addListener(ChangeListener<? super T>) void
removeListener(ChangeListener<? super T>) void
getValue() T
ChangeListener<T>
changed(ObservableValue<? extends T>, T, T) void
ReadOnlyProperty<T>
getBean() Object
getName() String
Binding<T>
isValid() boolean
invalidate() void
getDependencies() ObservableList<?>
dispose() void
WritableValue<T>
getValue() T
setValue(T) void
Property<T>
bind(ObservableValue<? extends T>) void
unbind() void
isBound() boolean
bindBidirectional(Property<T>) void
unbindBidirectional(Property<T>) void
Powered by yFiles

Figure 1

text. As you'll see when we walk through the relevant code, this demonstrates the concept of lazy evaluation.

Also, when you click the **Lap** button, the lap time at that moment is

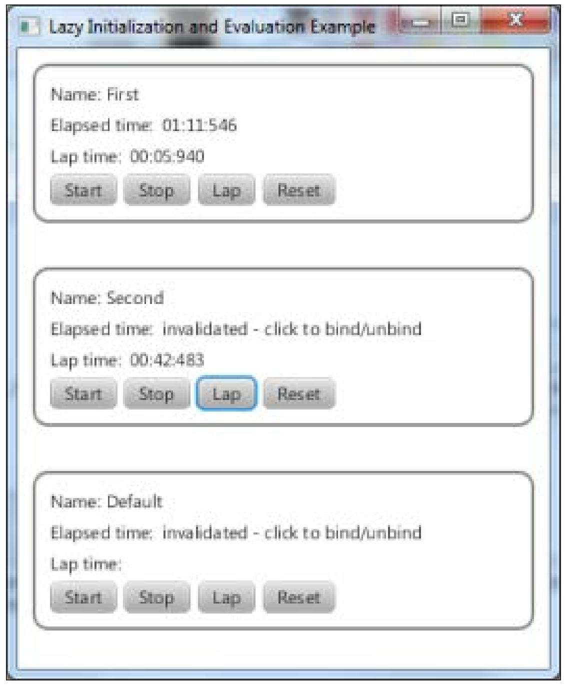

Lazy Initialization and Evaluation Example

Name: First
Elapsed time: invalidated - click to bind/unbind
Lap time:

Name: Second
Elapsed time: invalidated - click to bind/unbind
Lap time:

Name: Default
Elapsed time: invalidated - click to bind/unbind
Lap time:

Figure 2


Lazy Initialization and Evaluation Example

Name: First
Elapsed time: 01:11:546
Lap time: 00:05:940

Name: Second
Elapsed time: invalidated - click to bind/unbind
Lap time: 00:42:483

Name: Default
Elapsed time: invalidated - click to bind/unbind
Lap time:

Figure 3

displayed. You may also click the **Stop** button to cause the elapsed time to stop increasing, and you may click the **Reset** button to reset both the elapsed and lap times.

Obtaining and Running the LazyInitEvalExercise Project

1. Download the NetBeans project file, which includes the LazyInitEvalExercise program.
2. Expand the project into a directory of your choice.
3. Start NetBeans, and select **File -> Open Project**.
4. From the Open Project dialog box, navigate to your chosen directory and open the LazyInitEvalExercise project, as shown in **Figure 4**. If you receive a message stating that the jfxrt.jar file can't be found, click the **Resolve** button and navigate to the rt/lib folder subordinate to where you installed the JavaFX 2 SDK.
   Note: You can obtain the NetBeans IDE from the NetBeans site.
5. To run the application, click the **Run Project** icon on the toolbar, or press the F6 key. The icon looks like the Play button on a DVD player, as shown in **Figure 5**.

The LazyInitEvalExercise application should appear in a window, as shown in **Figure 6**.

The behavior of LazyInitEvalExercise is different in a few ways from LazyInitEvalSolution. For example, as shown in **Figure 6**, clicking the **Start** button and then clicking the invalidate text causes the elapsed time to be displayed as an integer, rather than in minutes:seconds:milliseconds format. This is because LazyInitEvalSolution uses a custom binding that you'll implement in one of the steps below.

Below are the steps you can follow to implement all of the behavior in LazyInitEvalSolution.

Step 1: Gain an Understanding of Lazy Versus Eager Binding Evaluation

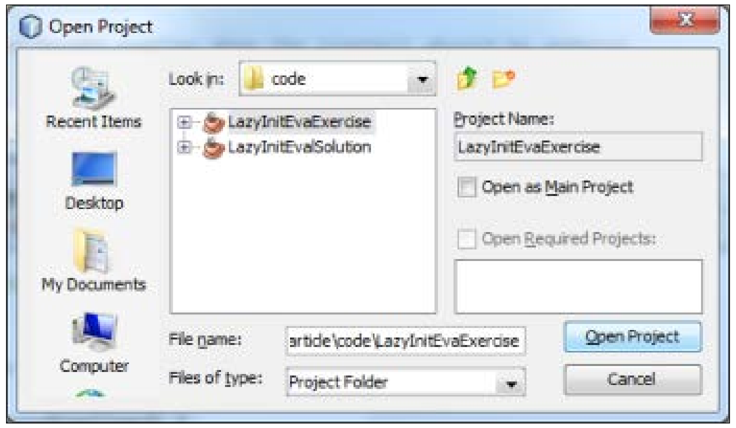
A binding can be evaluated in either an *eager* or a *lazy* manner:

- When a binding is evaluated in an eager manner, the updated value of the binding is calculated whenever the binding is invalidated.
- When a binding is evaluated in a lazy manner, the updated value of the binding is calculated only when needed.

Take a look at the code in the StopWatchNode.java file in the LazyInitEvalExercise project, which shows the starter code for this example. We'll show code snippets from LazyInitEvalMain.java as you perform the steps in this exercise.

Implementing lazy evaluation in a binding. As shown in **Listing 1**, some of the code in StopWatchNode.java instantiates a SimpleStringProperty, binds it to the elapsedMillisProperty property of the StopWatchModel class, and adds an InvalidationListener to it.

As a result, when the value of elapsedTimeStrProperty is no longer valid, it isn't evaluated immediately; hence, the term *lazy evaluation*.


Open Project

Look in: code
LazyInitEvaExercise
LazyInitEvalSolution
Project Name: LazyInitEvaExercise
Open as Main Project
Open Required Projects:
File name: article\code\LazyInitEvaExercise
Files of type: Project Folder

Figure 4

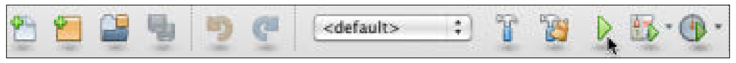
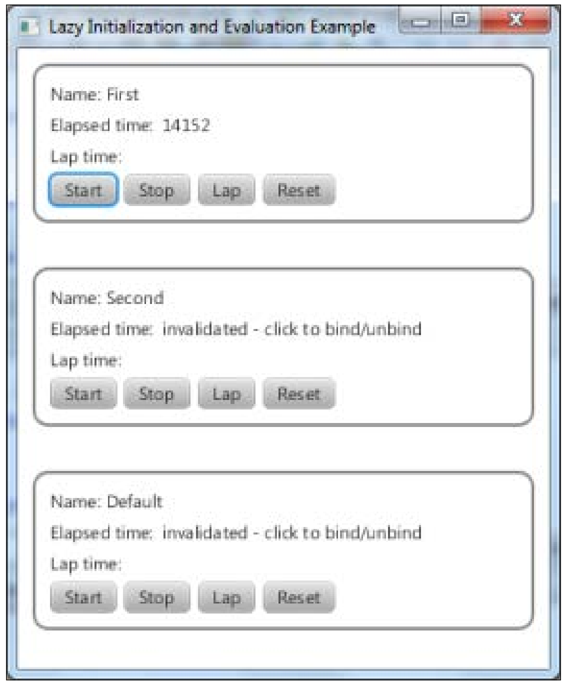


Figure 5


Lazy Initialization and Evaluation Example

Name: First
Elapsed time: 14152
Lap time:

Name: Second
Elapsed time: invalidated - click to bind/unbind
Lap time:

Name: Default
Elapsed time: invalidated - click to bind/unbind
Lap time:

Figure 6

Implementing eager evaluation in a binding. As shown in **Listing 2**, when the user clicks the invalidate – click to bind/unbind text, the textProperty of the elapsedNode is bound to the elapsedTimeStrProperty. This causes the binding to be evaluated in an eager manner, because the elapsedNode (a Label control) gets and displays the value of the elapsedTimeStrProperty whenever it is invalidated.

By using a lazy evaluation strategy when applicable, an application can avoid unnecessary processing, often improving performance as a result. Let's move on to another optimization technique: lazy initialization of properties.

BEING LAZY IS GOOD
By using a lazy evaluation strategy, an application can avoid unnecessary processing, often **improving performance.**

Step 2: Implement Lazy Initialization of a Property
A JavaFX property can be initialized in either an eager or a lazy manner. There are several patterns for implementing lazy initialization, the two most popular being *half-lazy* and *full-lazy*.

To see the eager initialization approach, take a look at **Listing 3**, in which you'll see three properties defined in the StopWatchModel.java file from the LazyInitEvalExercise project.

By convention, each of these properties has setter, getter, and property methods. For example, the lapMillis property shown in **Listing 3** has methods named setLapMillis(), getLapMillis(), and lapMillisProperty(). The implementation of these methods causes the SimpleIntegerProperty to be instantiated the first time that the lapMillis property is accessed via the setLapMillis(), getLapMillis(), or lapMillisProperty() method. This can be a waste of memory resources, depending on how the property is used in practice. Let's explore ways to optimize this, beginning with half-lazy initialization.

Implementing a half-lazy initialization approach. As shown in **Listing 4**, the half-lazy initialization approach avoids instantiating the SimpleIntegerProperty referenced by lapMillis when the getLapMillis() is called. It also avoids instantiation when the setLapMillis() method is called with the lapMillis property's default value, which is held in DEFAULT_LAP_MILLIS.

Go ahead and plug in the code from **Listing 4** into the StopWatchModel.java file, and we'll discuss a more aggressively lazy strategy.

Implementing a full-lazy initialization approach. As shown in **Listing 5**, the full-lazy initialization approach avoids instantiating the SimpleStringProperty referenced by name when the getName() method is called. It also avoids instantiation whenever the setName() method is called. It does the latter at the cost of the extra instance variable nameStr, but this is a good approach when the property won't typically be used in a binding.

Go ahead and plug in the code from

LISTING 1 / LISTING 2 / LISTING 3 / LISTING 4 / LISTING 5

```
private StringProperty elapsedTimeStrProperty = new SimpleStringProperty();

... code omitted ...

  elapsedTimeStrProperty.addListener(new InvalidationListener() {
    public void invalidated(Observable observable) {
      if (!elapsedNode.textProperty().isBound()) {
        elapsedNode.setText("invalidated - click to bind/unbind");
      }
    }
  });

  elapsedTimeStrProperty.bind(stopWatchModel.elapsedMillisProperty()
                .asString());
```

Download all listings in this issue as text

Listing 5 into the StopWatchModel.java file. When you're done, let's move on to the next concept, which involves creating a custom binding.

Step 3: Create a Custom Binding

As mentioned previously, the elapsed time value is formatted as an integer in **Figure 6** but as a minutes:seconds:milliseconds string in **Figure 3**. This is accomplished by using a custom binding instead of the default binding provided by the SimpleIntegerProperty.

To implement a custom binding, our example defines a class named TimeStringBinding that extends the StringBinding class, which implements the Binding interface shown in **Figure 1**. You can see the code for the TimeStringBinding class in **Listing 6**.

As shown in **Listing 6**, to implement the custom binding we do the following:

- Define a constructor with the input arguments for the binding, calling the bind() method of the superclass.
- Override the computeValue() method, returning the desired output value for the binding.

To use this custom binding in the LazyInitEvalExercise project, comment out the following lines in the StopWatchNode.java file:

```
elapsedTimeStrProperty.bind(
 stopWatchModel.elapsedMillisProp
erty()
                   .asString()
);
```

Then, add the following lines to StopWatchNode.java:

```
elapsedTimeStrProperty.bind(
 new TimeStringBinding(
  stopWatchModel.elapsedMillis
Property()
 )
);
```

For extra credit, also replace the binding for the lapNode textProperty() in a similar manner so that it appears as shown in **Figure 3**.

Go ahead and run the LazyInitEvalExercise application to verify that the custom bindings cause the elapsed time and lap time values to be formatted as shown in **Figure 3**.

Conclusion

JavaFX 2 comes with numerous classes and interfaces that provide a powerful properties and bindings framework. You can optimize the evaluation of bindings and the initialization of properties by using the lazy approaches demonstrated above. In addition, you can define custom bindings when augmenting the default behavior of the bindings in the JavaFX 2 API as desired. </article>

LEARN MORE

- "Using JavaFX Properties and Binding"
- Michael Heinrichs' blog, which includes entries on JavaFX properties and bindings

LISTING 6

```
package javafxpert.lazyiniteval.binding;

import java.text.SimpleDateFormat;
import javafx.beans.binding.StringBinding;
import javafx.beans.property.IntegerProperty;

public class TimeStringBinding extends StringBinding {
 private SimpleDateFormat timeFormat = new SimpleDateFormat("mm:ss:SSS");
 private IntegerProperty millis;

 public TimeStringBinding(IntegerProperty millisArg) {
  super.bind(millisArg);
  millis = millisArg;
 }

 @Override
 protected String computeValue() {
  int elapsedInt = millis.get();
  return timeFormat.format(elapsedInt);
 }
}
```

Download all listings in this issue as text

ADAM BIEN

BIO

Threading and Concurrency

Java EE 6 reflects that centralized management of scarce resources is becoming more important.

From the beginning, application servers were designed to provide shared resources to multiple applications at the same time. Scarce resources, such as CPU time, file systems, or RAM, are supposed to be shared between applications. Fair resource sharing, however, requires some cooperation from the deployed applications.

One of the programming restrictions introduced with the Enterprise JavaBeans (EJB) 1.0 specification was "An enterprise bean is not allowed to start new threads or attempt to terminate the running thread" (page 117). Current restrictions on popular cloud platforms are similar to the Java EE programming model. You could argue that application servers have been cloud-ready since the advent of EJB 1.0 in 1998.

Note: Download the source code for the sample application described in this article.

Forbidden Threads

Without explaining why, EJB 1.0 prohibited the direct use of threads and synchronization primitives. The EJB 3.1 specification is even stricter: "The enterprise bean must not attempt to manage threads. The enterprise bean must not attempt to start, stop, suspend, or resume a thread, or to change a thread's priority or name. The enterprise bean must not attempt to manage thread groups."

THREAD CONTROL

The uncontrolled creation of threads affects not only scalability and performance but also robustness.

The EJB 3.1 specification explains the reason behind the increased restriction: "These functions are reserved for the EJB container. Allowing the enterprise bean to manage threads would decrease the container's ability to properly manage the runtime environment" (page 599).

In practice, application servers can easily prevent direct Thread manipulation by activating the SecurityManager. In the java.lang.Thread constructor, the SecurityManager is asked for permission. However, most application servers operate without SecurityManager, so applications can easily violate the restrictions without any consequences.

In addition, the statement "Allowing the enterprise bean to manage threads would decrease the container's ability to properly manage the runtime environment" is way too mild. The uncontrolled creation of threads affects not only scalability and performance but also robustness. Each thread consumes memory. Having too many threads eventually leads to OutOfMemoryError occurrences and not only affects the stability of the application that starts the threads, but can crash the entire application server and all the deployed applications.

On the other hand, the application server reuses threads from managed and monitored pools. You can securely prevent OutOfMemoryError occurrences by regularly performing stress tests, as described in "Stress Testing Java EE 6 Applications," and using the results to configure the application server thread pools. A properly configured thread pool will reject new requests to prevent server overloading.

Web tier specifications, such as Java Servlet 3.0, are more liberal regarding thread management. There are no programming restrictions related to threading in the Servlet, Java API for RESTful Web Services (JAX-RS), or JavaServer Faces (JSF) specifications. Moving restricted threading code from the business layer to the presentation logic or Java Management Extensions (JMX) beans does not solve the problem at all. It's just a hack around the useful restrictions of the EJB specification. Usually, the Web container and EJB/Contexts and Dependency Injection (CDI) container are executed in the same Java Virtual Machine (JVM). Uncontrolled thread creation in the Web container can also break the application and crash the server.

PHOTOGRAPH BY THOMAS EINBERGER/ GETTY IMAGES

@Asynchronous

Before Java EE 6, there was no easy way to start threads legally. The Service Activator pattern (mis)used Java Message Service (JMS) to asynchronously execute synchronous EJB beans.

Although Service Activator is conceptually a Gang of Four (GoF) Decorator pattern, the J2EE realization was overly complicated. You had to encode a synchronous method call into a JMS message in a proxy implemented as an EJB bean. The JMS message was sent with a temporary "response queue" to a message-driven bean (MDB), which was asynchronously invoked by the JMS runtime. In the onMessage method, the JMS message was decoded and the actual EJB method was invoked asynchronously.

The MDB waited until the execution of the actual method, converted the return value into a JMS message, and sent the message back to the proxy. The implementation of this pattern didn't fit on a single page. The Service Activator pattern can be summarized as a local Remote Procedure Call over JMS.

With EJB 3.1 and Java EE 6, the whole Service Activator implementation can be replaced with a single annotation. To execute a void method asynchronously, annotate it with @Asynchronous:

```
@Stateless
public class LongTask {
 @Asynchronous
 public void execute(){
  //heavy lifting
 }
}
```

Asynchronous methods can also return an instance of java.util.concurrent.Future:

```
@Stateless
public class LongTask {
 @Asynchronous
 public Future<String> execute(){
  String result = ...;
// heavy lifting
  return new AsyncResult<String>
(result);
 }
}
```

The first sentence of the Javadoc for Future perfectly explains its responsibilities: "A Future represents the result of an asynchronous computation." A Future instance is returned immediately, and you can use the Future#isDone and Future#isCancelled methods to check the progress. You can also use the Future#get method to fetch the return value. If the task is not completed, the invocation of Future#get will block until the result is computed.

Batch Processing

Using @Asynchronous for void methods feels natural. Future, on the other hand, does not appear to be particularly useful in a standard request-response scenario. The method annotated with the @Asynchronous annotation immediately returns a Future instance, but the Future#get call blocks.

Instead of invoking the get method, you could use isDone to check the availability of the result, but that is not optimal either. The client would effectively burn CPU cycles.

Future is a useful tool for the parallelization of multiple task executions in batch mode. An additional stateless EJB bean is needed to execute the @Asynchronous methods, cache the Future in a List and, finally, gather the results, as shown in **Listing 1**.

The first loop in **Listing 1** submits chunks of work to the application server's thread pool. The execute() method immediately returns a Future instance, which is stored in a java.util.List. In the second loop, all Future instances are asked for their results through the Future#get call. Although the method Future#get blocks, in N-1 cases, the call will also return immediately.

The execution of the method BatchTask#executeBatch will take as long as the longest LongTask#execute computation.

The code looks sequential, but it isn't. In each iteration of the first loop, a task is passed to a worker pool to be executed asynchronously. The for-each loop iterates over Future instances and occasionally blocks if the task is not computed yet. However, during the wait time for a computation result, all other tasks are still executed in the background.

The Right Tool for Fire-and-Forget

@Asynchronous is a great tool for the implementation of fire-and-forget use

LISTING 1

```
@Stateless
public class BatchTask {
  @Inject
  LongTask lt;

  public void executeInBatch() {
    List<Future<String>> results = new ArrayList();
    //parallelize
    for(int i=0;i<10;i++){
      results.add(lt.execute());
    }
    //gather
    for (Future<String> resultProxy : results) {
      try {
      resultProxy.get(); // use the result, or not
      } catch (Exception ex) { }
    }
  }
}
```

Download all listings in this issue as text

cases. All submitted tasks are queued and executed with threads managed by the application server. The task queue is a transient data structure—in the event of an application server crash, all submitted, but not yet processed, tasks will get lost.

Because the task queue is transient, an @Asynchronous method cannot be used as a replacement for the Service Activator pattern implemented with persistent JMS queues.

If a transient thread queue is not an option, you can use persistent single-action timers for asynchronous invocation. In the executeAsync method exposed to the client, you have to register a persistent single-action timer first, as shown in **Listing 2**.

Even parameters can be passed to the timer as a Serializable instance. You have to wrap all parameters in a Serializable holder to pass them to the timer. Because the timer is configured to be persistent, the application server has to persist the information caller's transaction prior to the timer execution. No such overhead is needed for the @Asynchronous methods, which results in better performance.

Furthermore, the EJB 3.1 specification does not define any overload behavior for @Asynchronous invocations. The maximum number of concurrent requests—and, thus, threads—is unspecified. Most application servers allow you to configure in a proprietary way the thread pool dedicated for the execution of the @Asynchronous method.

Asynchronous Servlets

The Java Servlets 3.0 specification introduced with Java EE 6 supports asynchronous processing. From the HTTP client perspective, there is no difference between an asynchronous and a "traditional" Servlet request. In both cases, the client is blocked until the request is completed.

An asynchronous request allows the Web container to block the browser connection without binding a thread. The service method immediately returns, but the AsyncContext instance can still be used to send messages back to the browser or another HTTP client (for example, a JAX-RS client), as shown in **Listing 3**.

To perform an asynchronous request, the Servlet with all filters involved in the invocation chain has to be deployed with the asyncSupported=true option activated. The method HttpServletRequest#startAsync() puts the request into asynchronous mode and returns an AsyncContext instance immediately. The behavior is similar to an @Asynchronous method returning the Future instance. An AsyncContext instance effectively represents the browser window (or the HTTP client) and is used to communicate with the client after the completion of the service method (and doGet, doPost, and so on).

In **Listing 3**, the AsyncServlet sends the AsyncContext as a CDI event. The AsyncContext is received by the EventBroker singleton EJB bean shown in **Listing 4**.

The AsyncContext is stored in the lock-free CopyOnWriteArrayList contexts instance for future notifications. Each invocation of the EventBroker#onNewEvent method iterates over all AsyncContext instances stored in the contexts list, passes the String message parameter, commits the request with the AsyncContext#complete method and, finally, removes the AsyncContext from the list.

Each invocation of the onNewEvent method effectively distributes the String message parameter to all listening HTTP clients. The invocation of the onNewEvent method blocks until the message arrives at the client. It is a classic fire-and-forget call, so you can easily perform the call in the background by annotating the method with the @Asynchronous annotation. In the sample application, the String message event is fired in a JAX-RS resource (see **Listing 5**).

Usually, you would not want to expose the HTTP API directly to EJB beans

LISTING 2 / LISTING 3 / LISTING 4 / LISTING 5

```
@Stateless
public class PersistentAsynchronous {
  @Resource
  TimerService ts;
  public void executeAsync(String message){
    TimerConfig config = new TimerConfig(message, true);
    ts.createSingleActionTimer(1, config);
  }
  @Timeout
  public void execute(Timer timer){
   String message = (String)timer.getInfo();
   //do some work
  }
}
```

Download all listings in this issue as text

in the business layer and wrap the AsyncContext in a neutral wrapper. In the open source project LightFish, the class BrowserWindow is used to push GlassFish monitoring data to the JavaFX 2 client, as shown in **Listing 6**.

JCA—If You Want More

Java EE Connector Architecture (JCA) 1.6 connectors give you the most control over concurrency in a "legal," convenient, and monitorable way. Furthermore, JCA connectors are managed by the application server, which usually gives you many configuration options. Building a JCA 1.6 connector is not trivial, but it can be achieved in a few hours.

The implementation of the ResourceAdapter receives the BootstrapContext in the overridden start method, which gives you access to the WorkManager instance. You can safely ignore all other methods defined in the ResourceAdapter interface (see **Listing 7**).

You only need to compile the class shown in **Listing 7**, put it into a JAR file, change the .jar extension to .rar, and deploy the connector. The application server would create the WorkManagerBootstrap connector and pass the BootstrapContext with which the WorkManager instance can be obtained.

A WorkManager allows you to execute javax.resource.spi.work.Work instances that extend the Runnable interface. This solution would work, but your application would not be able to get a reference to the WorkManager without nasty hacks. However, this implementation would be sufficient to implement a manageable network server.

JCA connectors work in a connection-oriented way, so you will have to implement at least two additional interfaces: javax.resource.spi.ManagedConnection and javax.resource.spi.ManagedConnectionFactory. Fortunately, you can implement both interfaces once and reuse them for multiple JCA connector implementations.

The class ConnectionFactory (see **Listing 8a** and **Listing 8b**) exposes the meta-data to the runtime using the @ConnectionDefinition annotation and, together with the @Connector annotation on the WorkManagerBoostrap (see **Listing 7**), makes the Resource Adapter Archive (RAR) XML deployment descriptor obsolete.

The WorkManagerBootstrap is instantiated once at connector deployment time and is effectively a singleton. The ConnectionFactory needs an instance of the WorkManager stored inside the WorkManagerBootstrap.

JCA 1.6 SPI comes with a useful interface, javax.resource.spi.ResourceAdapterAssociation, which is used by the container to associate the ResourceAdapter implementation (WorkManagerBootstrap) with the ManagedConnectionFactory implementation. After casting the ResourceAdapter to WorkManagerBootstrap, the ConnectionFactory can access the WorkManager instance that is needed for asynchronous execution.

The ConnectionFactory is just the necessary plumbing for managing and pooling the connections properly. The

LISTING 6 / LISTING 7 / LISTING 8a / LISTING 8b

```
public class BrowserWindow {

  private AsyncContext asyncContext;
  private final ServletResponse response;
  private String channel;

  public BrowserWindow(AsyncContext asyncContext) {
    this.asyncContext = asyncContext;
    this.response = this.asyncContext.getResponse();
    this.response.setContentType("application/xml");
  }

  public BrowserWindow(AsyncContext asyncContext,String channel){
    this(asyncContext);
    this.channel = channel;
  }

  public void send(){
    try{
      this.asyncContext.complete();
    }catch(Exception e){   }
  }

  public Writer getWriter(){
    try {
      return this.asyncContext.getResponse().getWriter();
    } catch (IOException ex) {
      throw new IllegalStateException("Cannot return writer: " + ex,ex);
    }
  }

  public void nothingToSay(){
    HttpServletResponse httpServletResponse = (HttpServletResponse) response;
    httpServletResponse.setStatus(204);
    this.asyncContext.complete();
  }

  public String getChannel() {
    return channel;
  }
}
```

Download all listings in this issue as text

method setMaxNumberOfConcurrentRequests exposes a writable property to the application server's administration or configuration facilities. Pooling of already created ManagedConnection instances requires implementation of the matchManagedConnections method.

Usually, a connection is differentiated by the username, password, or other security credentials. Here, all thread pools are equal, so we just match the connection using the ConnectionRequestInfo instance.

Both the ManagedConnection implementation (Connection) and the ManagedConnectionFactory (ConnectionFactory) realize connection management at the logical level and are highly reusable. In addition, the Connection class fires events to notify the application server runtime about the current state (see **Listing 9a**, **Listing 9b**, and **Listing 9c**).

The Connection instance manages a List of ConnectionEventListeners and fires events on close, commit, begin, or rollback events. Our connector implementation is not transactional (but could be implemented as such), so most of the events will never fire.

As a logical part, the Connection also manages its physical counterpart. JCAExecutor is the actual "meat" of the connector. Runnable instances are passed to the execute method to be asynchronously executed by the WorkManager (see **Listing 10**).

EVEN BETTER

Java EE 7 will come with even more interesting concurrency features.

The JCAExecutor implements the java.util.concurrent .Executor interface, which will be directly used by the application and is created by the JCAExecutorFactory, as shown in **Listing 11**.

JCAExecutor with its JCAExecutorFactory are the actual domain-specific implementation. Most of the ManagedConnection and ManagedConnectionFactory implementations are business logic–agnostic and could be reused for other JCA implementations. All classes have to be packaged into a JAR file that has the .rar extension and deployed to the application server.

Java EE applications do not care about the JCA implementation and are interested only in the asynchronous execution of tasks in a managed environment. The application-facing API consists of a single interface that returns the Executor to the application, as shown in **Listing 12**.

Now Java EE 6 applications can access a managed Executor implementation as easily as in a Java SE environment (see **Listing 13**).

Java EE 7—Even Better

With five classes and a simple Maven Project Object Model—and without any XML deployment descriptors—you get a flexible thread pool implementation that is fully managed by the application server and could be extended to support transactions, security, and progress monitoring.

LISTING 9a / LISTING 9b / LISTING 9c / LISTING 10 / LISTING 11

```
public class Connection
    implements ManagedConnection {

  private ConnectionFactory mcf;
  private PrintWriter out;
  private JCAExecutor fileConnection;
  private ConnectionRequestInfo connectionRequestInfo;
  private List<ConnectionEventListener> listeners;

  Connection(PrintWriter out,ConnectionFactory mcf, ConnectionRequestInfo
connectionRequestInfo) {
    this.out = out;
    this.mcf = mcf;
    this.connectionRequestInfo = connectionRequestInfo;
    this.listeners = new LinkedList<ConnectionEventListener>();
  }

  public Object getConnection(Subject subject, ConnectionRequestInfo
connectionRequestInfo)
      throws ResourceException {
    fileConnection = new JCAExecutor(out,this,mcf, connectionRequestInfo);
    return fileConnection;
  }

  public void destroy() {
    this.fileConnection.destroy();
  }

  public void cleanup() {
  }

  public void associateConnection(Object connection) {
    this.fileConnection = (JCAExecutor) connection;

  }
```

Download all listings in this issue as text

LISTING 12 LISTING 13

```
import java.io.Serializable;
import java.util.concurrent.Executor;
import javax.resource.Referenceable;

public interface WorkExecutorFactory extends Serializable, Referenceable {
    Executor newExecutor();
}
```

Download all listings in this issue as text

Java EE 7 will come with even more interesting concurrency features. JMS 2.0 is going to vastly simplify the process of sending and receiving messages. EJB 3.2 will likely support additional quality-of-service features, such as @MaxConcurrency. Java EE 7 will probably include a dedicated concurrency specification.

Threads managed by an application are not visible to the application server runtime. It is not possible to throttle the concurrency or monitor the application's behavior. Manually created threads not only threaten scalability, but they also massively affect the robustness of the overall system. The more cloud-ready your application needs to be, the more important central management of scarce resources and, thus, threads becomes. </article>

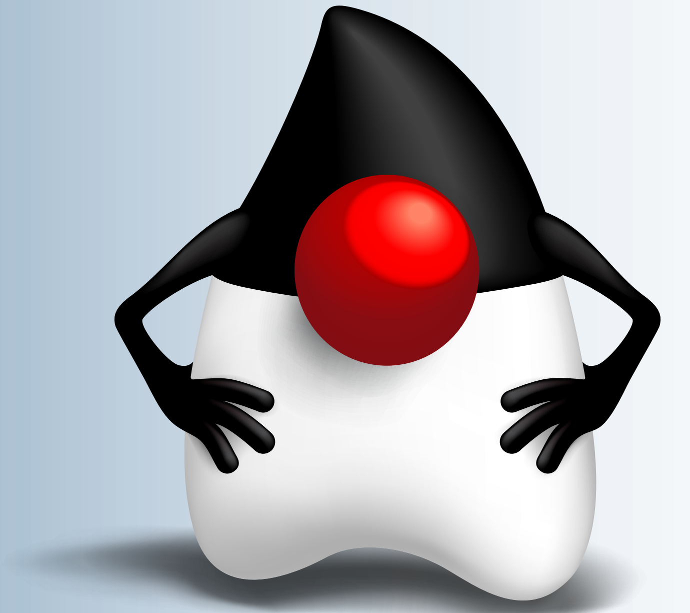
LEARN MORE

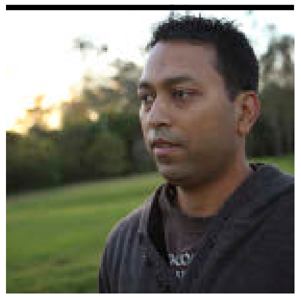
- connectorZ
- *Real World Java EE Night Hacks—Dissecting the Business Tier*, page 70 (press.adam-bien.com, 2011)

Java™

YOUR LOCAL JAVA USER GROUP NEEDS YOU

Find your JUG here





Wirelessly Back Up Your Device's Address Book

Learn to create a MIDlet to schedule a backup of your address book—and more.

VIKRAM GOYAL

BIO

PHOTOGRAPH BY JONATHAN WOOD/ GETTY IMAGES

Java ME applications are notoriously hard to back up. In this article, I will go through the process of creating a MIDlet application that will allow you to select a destination within your Bluetooth network, create a schedule to back up your address book, perform an on-demand backup, and log (and view) all of this activity. You can, in addition, use this MIDlet to extend backup activities to cover other data that your device allows you to access.

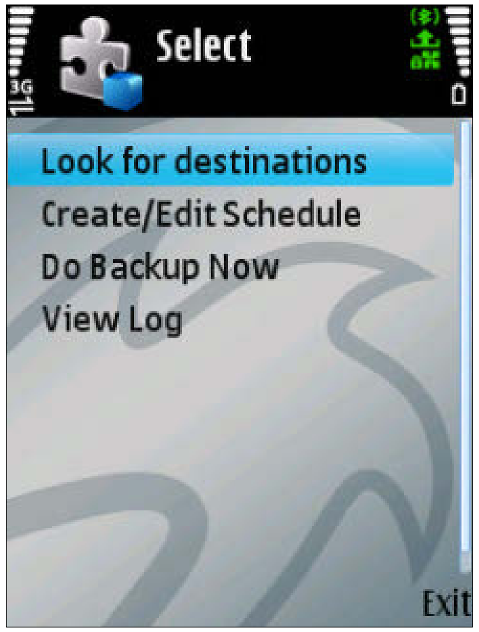


Figure 1

Note: The source code for the application described in this article can be downloaded as a NetBeans project here.

The Application Flow

Figure 1 shows the target application in action.

The application flow is divided among the four options, which represent logical activities. You can look for destinations within the device's Bluetooth environment, create and edit a schedule to control when the backup should be run automatically, perform an on-demand backup, and view the log.

Under normal circumstances, you would create a schedule, set up the destination once, and then not worry about anything else. In reality, only a signed application with the proper permissions would be able to do these things, because you would need to grant the application permission to read data and initiate transfers (both on the device and on the target platform).

I discuss the application flow in terms of these logical modules next by covering them one by one.

BACKUP PLAN

You can, in addition, use this MIDlet to extend backup activities to cover other data that your device allows you to access.

Looking for Destinations

I initiated the development of this MIDlet with the intention of using Wi-Fi to detect devices. However, even though the File Transfer API (JSR 75) supports the discoverability and manipulation of folders on remote devices, none of the implementation APIs (including Oracle's) supports this. I used Bluetooth instead to discover devices and the services they allowed.

I described the Bluetooth discovery process and how we can use it to send files over the network in a previous article, but I will cover the main points again here.

The basic code from that previous article hasn't changed much because the process of sending the data is pretty much the same.

What is different is that we need to search for a Bluetooth connection point, find the OBEX push profile, and *then store that information* so the MIDlet can reuse it without any more input from you.

To do so, we create a RecordStore and hold this connection point (the actual connection URL) in that database. This RecordStore is called the Backup Store in the code in **Listing 1**, and it is initiated when the MIDlet starts.

If this destination is not set, any attempt to create a backup will result in a simple message stating that a backup destination is not set.

The actual backup destination is set after the discovery of a location, as shown in the code in **Listing 2** (from the BluetoothServices class).

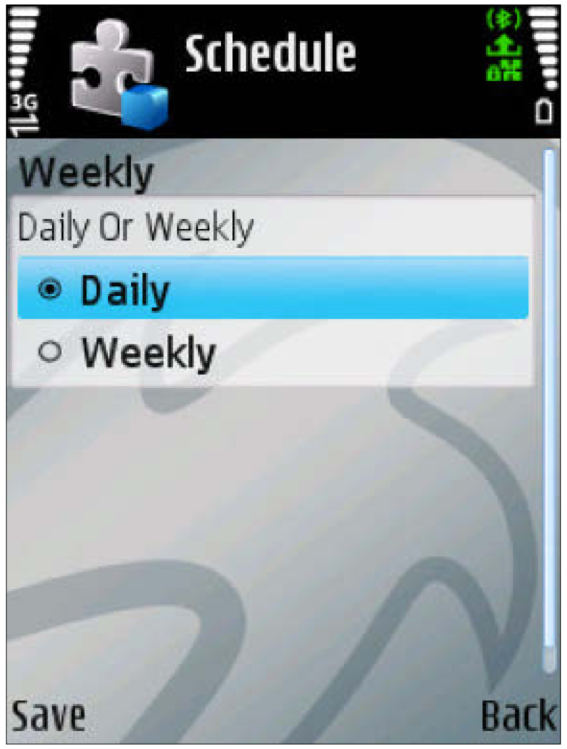
Note that we store the actual connection URL so that we don't need to discover the device or the services each time the backup needs to run. You can, of course, look for new destinations and store them if preferred.

The rest of the code isn't much different from my previous article. BluetoothServices implements the DiscoveryListener interface, so the deviceDiscovered, servicesDiscovered, serviceSearchCompleted, and inquiryCompleted methods all have a concrete implementation with related UI updates.

EASY ADDS

I have backed up only the address book. It would be easy to add a to-do list, a calendar list, or any other data that the MIDlet may access.

The doSend() method is invoked when the MIDlet wants to send compressed data to the destination. (There is more information about compression later in this article). The doSend() method accepts the destination URL, the filename that we want to use for this backup, and a ByteArrayOutputStream that holds the actual data.



Weekly
Daily Or Weekly
(●) Daily
( ) Weekly

Figure 2

Creating and Editing a Schedule

To keep this part simple, you have the option of selecting whether the backup should run automatically on a daily or weekly basis. Adding the ability to specify a time is easy enough and is left as an exercise for you.

Figure 2 shows the UI for setting the schedule.

The top of the screen shows what the current schedule is set to, and below that you can select whether to change the schedule. Selecting the **Save** UI option stores this information in another RecordStore called the Schedule Store.

The following code shows how we can save the schedule into our Schedule Store and then set up an alarm for self-invocation of the MIDlet.

This is where I use the registerAlarm() method of PushRegistry to set an alarm to restart the MIDlet at the next scheduled run, which is calculated by taking into account that there are 86,400,000 milliseconds in a day and 604,800,000 milliseconds in a week. Note that this is

LISTING 1 / LISTING 2 / LISTING 3 / LISTING 4 / LISTING 5

```
// load up the backup dest record store
rs = RecordStore.openRecordStore("Backup Store", false);
// this will get the last record--the last destination for our Bluetooth Device
backupDest = new String(rs.getRecord(rs.getNumRecords()));
if(backupDest == null) throw new RecordStoreNotFoundException("");
backupDestSet = true;
```

Download all listings in this issue as text

set to run in a separate thread to avoid any deadlock issues.

When the MIDlet automatically starts at the next scheduled run, the startBackup() method is called at the end of the constructor. This way, the MIDlet is launched automatically and starts the backup. In the constructor, I also re-register the next time the MIDlet should run based on the nextRun variable picked up from the Schedule Store where it is stored, as shown in **Listing 3**.

Viewing the Log

Before we get to the actual backup, let's look at the **View Log** UI option. Quite simply, it stores its data at certain crucial points in the MIDlet cycle in yet another RecordStore, the Log Store, as shown in **Listing 4**.

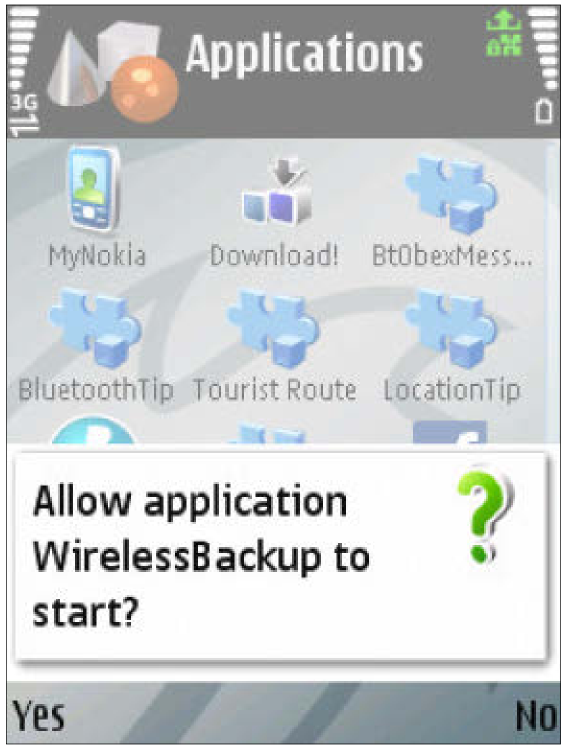
At each call to the log(String message) method, a time stamp is added to the message and stored in the log, as shown in **Listing 5**.

I have also provided an option to clear the log, because it can get fairly large. Selecting the **Clear Log** UI option simply deletes anything in the Log Store:

```
lrs.closeRecordStore();
RecordStore.deleteRecordStore("Log
 Store");
```

Creating a Backup

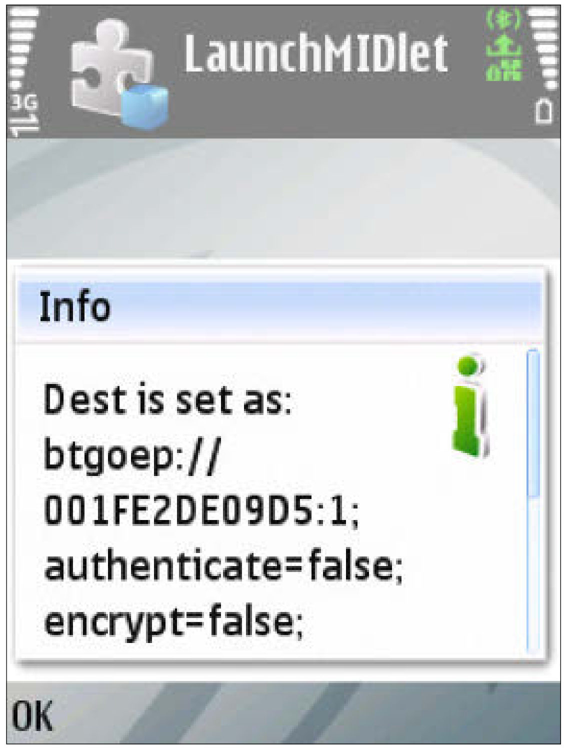
The actual creation of the backup happens in two methods: startBackup() and



Allow application WirelessBackup to start?

Yes No

Figure 3

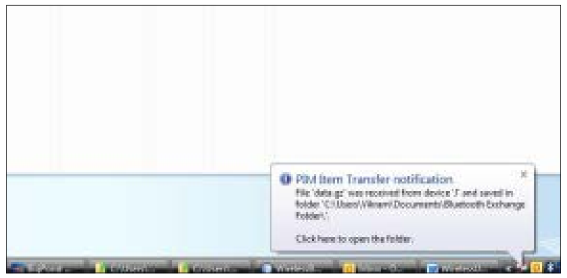


Info

Dest is set as: btgoep://001FE2DE09D5:1;authenticate=false;encrypt=false;

OK

Figure 4



PIM Item Transfer notification

File 'data.gz' was received from device 'J' and saved in folder 'C:\Users\Vikram\Documents\Bluetooth Exchange Folder\'.

Click here to open the folder.

Figure 5

LISTING 6 LISTING 7

```
new Thread() {
  public void run() {
    try {
      log("Backup Started At: " + new Date());
      ByteArrayOutputStream baos = new ByteArrayOutputStream();
      LZCOutputStream os = new LZCOutputStream(baos);
      // the address book; it's easy to add other lists
      ContactList addressbook =
        (ContactList)(PIM.getInstance().openPIMList(
          PIM.CONTACT_LIST, PIM.READ_ONLY));
      createBackup(addressbook, PIM.CONTACT_LIST, os);
      btServices.doSend(backupDest, "data.gz", baos);
      msgAlert.setString("Done");
      display.setCurrent(msgAlert, list);
      log("Backup Finished At: " + new Date());
    } catch (Exception ex) {
      ex.printStackTrace();
    }
  }
}.start();
```

Download all listings in this issue as text

createBackup(), although the latter is private and called only via the startBackup() method.

The startBackup() method is called either automatically when the MIDlet starts (via your initiation or via the PushRegistry alarm) or when you select the Do Backup Now UI option.

As expected, the startBackup() method first checks to see whether a backup destination has been defined. If the destination has not been defined, it alerts you that the backup can't start. Next, it creates a thread to do the backup in a manner that will not block the main thread. See **Listing 6**.

In the code in **Listing 6**, I have backed up only the address book. It would be easy to add a to-do list, a calendar list, or any other data that the MIDlet may access (which, admittedly, is limited and heavily implementation dependent).

To make the data transfer faster, I used an external library to compress the data before it is sent over the Bluetooth channel. This library, called compress-j2me, is extremely small and provides good compression.

I start by creating two output streams: one for holding the data and the other to compress it. The compression stream takes the first stream as the input and compresses it as it goes along. The actual looking up of the data is done in the createBackup() method.

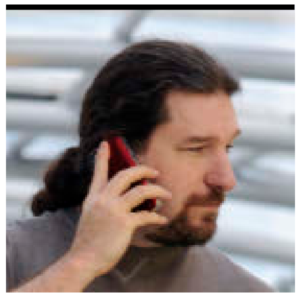
The code in **Listing 7** goes through the items in the address book, looks up a valid data_format for serializing the items (provided via the PIM API) and iterates over individual items, while adding these items to the compression stream. At the end, the stream is flushed and the data written out.

Once the createBackup() method is finished, the startBackup() method uses the doSend() method of the BluetoothServices class we discussed earlier to actually send the data to the connected device.

If you have established a trusted and paired connection between your device and its backup destination, you will get the data in a pre-established Bluetooth Exchange folder. I have tested this process with a live device (Nokia N95) and confirmed that it works perfectly. See **Figure 3**, **Figure 4**, and **Figure 5**, which show the activation of the PushRegistry, the destination confirmation, and data transfer confirmation on the target device. </article>

LEARN MORE

- "Discovering Devices and Sending Files via Bluetooth in Java ME"
- The compress-j2me project



TED NEWARD

BIO

PHOTOGRAPH BY
PHIL SALTONSTALL

Part 1

Oracle Berkeley DB Java Edition's Java API

Learn how the Java API for Oracle Berkeley DB Java Edition works.

In the summer of 2005, while at a Microsoft conference, I was asked what I thought of the then-current technical fad, the object-relational mapping (ORM) tool. My response, predictably, got some attention. I said that ORM is the Vietnam of computer science.

At the time, I was definitely "swimming upstream" because "everybody" knew that the only reasonable place to store data was in a relational database, preferably a big one. In fact, the bigger, the better—who wouldn't want more RAM, more disk space, faster I/O, the works.

I argued, in part, that no matter how much we try, the different "shapes" of the data—for example, our graphs of objects and the database's rectangular tables—create mismatches that are very hard to eliminate, no matter how sophisticated the tooling or libraries. It's particularly gruesome to try to shove a hierarchy into a table, for example, and trying to capture a cyclic graph (such as the one we see with the Facebook platform) is nearly impossible. Not that it can't be done, but when a subject requires an entire book on just that one idea (Joe Celko's *Trees and Hierarchies in SQL for Smarties*), there's probably more work there than we'd really like.

As the NoSQL movement gained prominence, developers realized that perhaps the relational database isn't the only place data needs to go. It might be the *last* place that data ends up, because that's where we have the wonderful tools for analysis, reporting, and transformation, but that doesn't mean it's the *only* kind of storage system available.

For decades, other storage systems have been quietly handling data quite nicely, efficiently, and compactly, and one of those, Berkeley DB, has been a staple for UNIX-based systems for years. Oracle acquired Berkeley DB a number of years ago and has made a Java API available for it.

Using Oracle Berkeley DB Java Edition is quite straightforward, partly due to its surface similarities to the relational model, but make no mistake—attempts to use it as such are likely to result in serious failure.

BEYOND RDBMS
As the NoSQL movement gained prominence, developers realized that **perhaps the relational database isn't the only place data needs to go.**

Essentials

Unlike many database APIs, the Java API for Oracle Berkeley DB Java Edition has two "levels."

The first level, the base API, is a low-level read/write set of operations that essentially thinks of everything in terms of key/value pairs, where the values are byte arrays that Java developers know how to create and parse, typically using either standard Java serialization or Oracle Berkeley DB Java Edition's own "bind" APIs.

As a general rule, unless you want or need to operate at this level of storage (and for the most part, the only reason you would want to do that is to be able to support duplicate primary keys), it's useful only to know that such accessibility is available and move on to the second API.

Developers who remember the C-based Berkeley DB APIs fondly will probably find themselves drawn to the base API, but be warned: the Oracle Berkeley DB Java Edition manual clearly states that the file format is not the same format used by the C edition, so files aren't portable between each API. Oracle Berkeley DB Java Edition files are portable across different Java platforms, however.

The second level, called the Direct Persistence Layer (DPL) API, is a more object-centric approach to data storage and more approachable for developers who are familiar with ORM tools, such as the Java Persistence API (JPA). However, it's important to note that Oracle Berkeley DB Java Edition isn't an ORM, because there aren't any actual tables in the storage files. Instead, the objects go directly into the database, without stopping to be pounded into rectangles first.

Exploration Testing
When working with a new API, one approach I like to take is called *exploration testing*. Essentially, I write unit tests, but not to test the product in question. Instead, exploration tests serve several different purposes:

- They provide a simple framework in which to try a number of things.
- They keep the exploration focused on small incremental progress.
- They let me make assertions first and then see if those assertions hold.
- And, most importantly, when a new version of the product comes out, running the tests against the new version tells me if anything has changed.

Getting Started
A general rule of thumb is that before any data can be stored or retrieved from a database, the program has to connect to (or "open") the database. In a traditional client-server RDBMS scenario, "opening" the database consists of providing networking information and credentials to (typically) open a TCP/IP socket to the database and authenticate against it. However, in an embedded database such as Oracle Berkeley DB Java Edition, opening the database usually consists of telling the API where to find the database on disk and whether to create the database if it doesn't exist.

In the Oracle Berkeley DB Java Edition API, doing so consists of creating an Environment object to represent the database environment. This requires a Java File object, indicating the directory into which the data files will go, which must exist ahead of time, plus an EnvironmentConfig object, which tells Oracle Berkeley DB Java Edition how it should behave under particular conditions.

For example, in order to keep the tests clean, the exploration tests should create and destroy the database each time, which means that EnvironmentConfig will need to have its allowCreate property set to true, as shown in **Listing 1**. Then, as can be inferred from **Listing 1**, closing the environment is done using the close() call.

Assuming that someSubdir exists in the directory in which the exploration tests are run, glancing inside reveals not just one file but several. Of the files there, the critical ones are suffixed with .jdb, and they are incrementally increasing numeric files. The first will be 00000000.jdb, and this is where the data will be written as a series of appending writes one after another.

In fact, the Oracle Berkeley DB Java Edition manual takes care to note that backing up these files doesn't require closing the database, as long as the backup process copies these files in order, starting from 0 and working upward.

LISTING 1

```
@Test public void openAndCloseADatabase()
 throws DatabaseException
{
 EnvironmentConfig config = new EnvironmentConfig();
 config.setAllowCreate(true);
 Environment dbEnv =
  new Environment(new File("./someSubdir"), config);

 dbEnv.close();
}
```

Download all listings in this issue as text

EntityStore
Once the database environment is opened, an EntityStore must also be opened on top of Environment in order to work with the DPL API. (Remember, DPL is an abstraction on top of the underlying key/byte-array-value store.)

Similar to how the Environment uses an EnvironmentConfig to describe configuration options for the Environment, the EntityStore is created with an EntityStoreConfig. And, just as EnvironmentConfig has to be told it's OK to create the files if they don't exist, the same is true for EntityStoreConfig, and this is done, again, through the allowCreate property.

All of this (and just about everything

else to do with the DPL API) comes out of the com.sleepycat.persist package.

For future tests, this code is moved to an @Before-annotated openDatabase method and an @After-annotated closeDatabase method, as shown in **Listing 2**.

Notice that I deliberately chose to delete the directory and re-create it each time just to ensure that no data files are lingering after each test. Obviously, for a production system, the logic would be different.

Entities

Just as with the JPA or any other object-based storage system, when working with the DPL API, we store and retrieve instances of objects.

Oracle Berkeley DB Java Edition knows that these objects are to be persisted because the class is annotated with the @Entity annotation. In addition, at least one field on the class must also be annotated with the @PrimaryKey annotation, whose purpose seems obvious, but whose actual use is different than in some other ORM systems.

For this example, let's assume the system is a blog engine (see **Listing 3**).

While it might appear that we could store a new blog post to the EntityStore through some kind of "store" method on the EntityStore, it's not that simple. The Oracle Berkeley DB Java Edition engine insists that we pay closer attention to the indexes defined within the store. The most obvious index is the one that keys off the @PrimaryKey for a given type.

To store or retrieve an object by its primary key, we first have to obtain that index (a PrimaryIndex<K,V> type, where K is the PrimaryKey-annotated type and V is the Entity-annotated type that owns it) from the EntityStore.

Once that's done, we can use the PrimaryIndex to put objects into the EntityStore and use get to retrieve objects from the EntityStore by the PrimaryKey, as shown in **Listing 4**.

Nary an SQL statement is found, which is great, assuming that the exact object (or at least its primary key value) is known to the code doing the lookup for the object. Sometimes, though, the entire list of objects (such as a list of the last 10 blog posts) must be looked at, in which case an entity iterator is needed, which is, again, retrieved from the PrimaryIndex<> object (see **Listing 5**).

Take very careful note of the close() call on the cursor at finish—failing to do so will yield an exception from the database runtime when the database itself is closed. (This is best dropped into a finally block, but because these are just exploration tests, we can live with it the way it is for now.)

For those cases where the primary key needs to be an artificial key, such as in a monotonically increasing numeric sequence, Oracle Berkeley DB Java Edition's API can generate that key automatically by modifying the @PrimaryKey annotation to use a "sequence," for example:

```
private @PrimaryKey(sequence=
"Sequence_Namespace") int id;
```

LISTING 2 / LISTING 3 / LISTING 4 / LISTING 5

```
public class BDBTest
{
 // ...

 File dbDir = new File("./data");
 Environment dbEnv;
 EntityStore dbStore;
 @Before public void openDatabase()
 {
  if (!dbDir.exists())
   dbDir.mkdir();

  EnvironmentConfig config = new EnvironmentConfig();
  config.setAllowCreate(true);
  dbEnv = new Environment(dbDir, config);

  StoreConfig storeConfig = new StoreConfig();
  storeConfig.setAllowCreate(true);
  dbStore = new EntityStore(dbEnv, "EntityStore", storeConfig);
 }
 @After public void closeDatabase()
 {
  dbStore.close();

  dbEnv.close();

  dbDir.delete();
 }
}
```

Download all listings in this issue as text

This is equivalent to using globally unique identifiers (GUIDs) as your primary key—keys that are entirely opaque and for which the value is irrelevant to the actual contents of the object. Any ability to browse objects and fetch them directly by primary key is lost, but nasty business problems are avoided when using mutable state for primary keys.

Secondary Keys

While this is great, sometimes we need to fetch objects by a criterion other than the primary key. For example, in a blog system, the blog often needs to show the blog entries for a given day, rather than by their title. This means that the database has to pony up the objects via a lookup scheme other than the primary key.

In Oracle Berkeley DB Java Edition, this is done by annotating fields with @SecondaryKey and retrieving them from the EntityStore by first obtaining a SecondaryIndex instance, as shown in **Listing 6**.

The SecondaryIndex<> type is tied to the PrimaryIndex<> type directly. The first generic parameter is the type of the SecondaryKey-annotated field; the second generic parameter is the type of the PrimaryKey-annotated field; and the third type is the Entity type itself. The constructor similarly uses the index, the type of the SecondaryKey-annotated field, and the name of that field.

And if the system needs to fetch a "range" of objects along that SecondaryIndex, two example values (one "from" and one "to") need to be passed in to the entities() call, along with a Boolean for each indicating whether that example value should be included in the returned set (true means inclusive and false means exclusive). See **Listing 7**.

By the way, the SecondaryKey instances can have a variety of different relationships to their values, as expressed by the relate parameter in the annotation description. The Javadoc contains the details, but essentially it's the usual 1:1, 1:n, n:1, or m:n relationships seen in every other database system on the planet. Because multiple blog posts can be posted on a single day, the postingDate field in the BlogPost type is annotated appropriately:

```
private @SecondaryKey(relate=
MANY_TO_ONE) Date postingDate;
```

Note that the MANY_TO_ONE is a Relationship enumeration, so for this to work, that Relationship type must be imported statically; otherwise, the code needs to read Relationship.MANY_TO_ONE, as usual.

Wrapping Up . . . for Now

In Part 2, there will be more to explore on how to store more-complex objects, but this article provided enough to get a feel for how Oracle Berkeley DB Java Edition's DPL API works. It's essentially an object persistence store, instead of an object-relational persistence store, and this means that there is far less administrative work in order to persist objects.

In fact, compared to a relational schema, there's effectively no work whatsoever: no data definition language, no server instances to stand up, no authentication credentials to establish, no authorization rules, nothing. Just create a directory, and away we go!

One caveat: As future enhancements are made to the BlogPost type (perhaps we will add an Author type, describing the author of the blog post for those blog systems that are multi-tenanted, such as WordPress), Oracle Berkeley DB Java Edition will try to just roll with the changes that occur in those types, thus allowing you to refactor additively (meaning add-only kinds of refactorings, for example, adding fields) without requiring any change to code. But changes such as adding the SecondaryKey annotation to the postingDate field will require a more hands-on approach to evolve existing data stores.

LISTING 6 LISTING 7

```
@Test public void storeAndRetrieveOneByDate()
{
 BlogPost newPosting =
  new BlogPost("The Vietnam of Computer Science");

 PrimaryIndex<String,BlogPost> primaryIndex =
  dbStore.getPrimaryIndex(String.class, BlogPost.class);
 primaryIndex.put(newPosting);

 SecondaryIndex<Date, String, BlogPost> dateIndex =
  dbStore.getSecondaryIndex(primaryIndex, Date.class,
   "postingDate");

 EntityCursor<BlogPost> postCursor =
  dateIndex.entities();
 for (BlogPost post : postCursor)
 {
  assertTrue(post.getTitle().contains("Computer Science"));
 }
 postCursor.close();
}
```

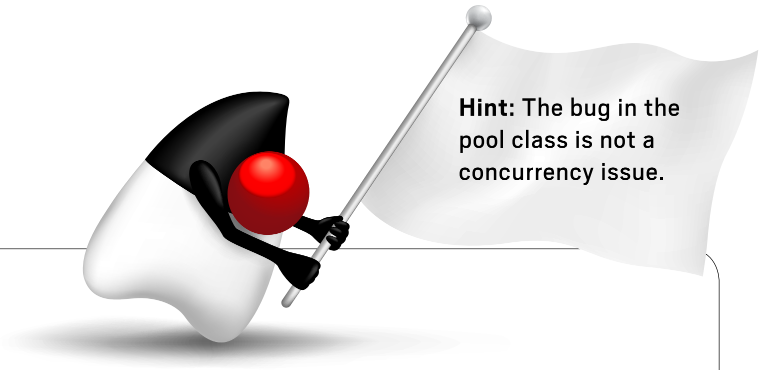
Download all listings in this issue as text

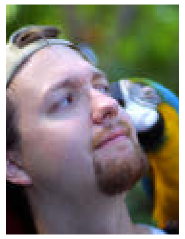
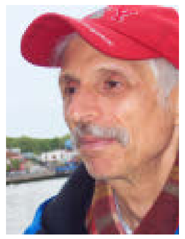
For exploration tests, this isn't a big deal, particularly because I'm blowing away the database entirely after each test. But for real databases, this could present a problem between version 1 and version 2. Oracle Berkeley DB Java Edition calls this evolution *mutation*, and while the database can handle some of the kinds of mutation that occur during refactoring, complex cases might require you to write utilities to migrate databases by hand.

Finally, I am not advocating that every relational database instance known to humanity be completely and utterly destroyed and replaced by Oracle Berkeley DB Java Edition, Apache CouchDB, or anything else.

Stay tuned for Part 2, where things will get a little more complex. </article>

//fix this /


Hint: The bug in the pool class is not a concurrency issue.

In the May/June 2012 issue, Angela Caicedo presented us with a JavaFX code challenge around binding. She asked us to consider a code fragment and determine the output.

The correct answer is #4: you will get an exception when you try to call the set method on myBoundInt. What is happening? myBoundInt is a bound variable, and it is not supposed to be changed directly. You can only change the value of myBoundInt through myInt, the property myBoundInt is bound to.

This issue's challenge comes from Jason Hunter (top left), author of *Java Servlet Programming*, 2nd Edition (O'Reilly Media) and deputy CTO at MarkLogic, and Boris Shukhat, applications programming manager, vice president, Bank of America Merrill Lynch.

1 THE PROBLEM

The ConnectionPool.java example from Chapter 9 of *Java Servlet Programming* has a subtle bug. The pool code worked fine for many years, but a recent upgrade to a JDBC driver broke it.

2 THE CODE

Below is a trimmed code snippet from the pool class. Can you spot the problem and fix it in place without redesigning the whole program?

```
  private Hashtable connections = new Hashtable();
  private void initializePool(...) ... {
    for (int i = 0; i < initialPoolSize; i++)
connections.put(getNewConnection(...), Boolean.FALSE); //
false=free
    }
  }
  public Connection getConnection() ... {
    // ... find a connection with a FALSE flag ...
    connections.put(con, Boolean.TRUE);
    return con;
  }
  public void returnConnection(Connection returned) {
    connections.put(returned, Boolean.FALSE);
  }
```

3 WHAT'S THE FIX?

1) Instead of Connection, use a PooledConnection.
2) Instead of Hashtable, use another implementation of the Map interface.
3) Extend the problematic Connection implementation, overriding some methods.
4) Create an implementation of Connection wrapping the problematic Connection.

GOT THE ANSWER?
Look for the answer in the next issue. Or submit your own code challenge!

ART BY I-HUA CHEN

EDITORIAL
Editor in Chief
Justin Kestelyn
Senior Managing Editor
Caroline Kvitka
Community Editors
Cassandra Clark, Sonya Barry, Yolande Poirier
Java in Action Editor
Michelle Kovac
Technology Editors
Janice Heiss, Tori Wieldt
Contributing Writer
Kevin Farnham
Contributing Editors
Blair Campbell, Claire Breen, Karen Perkins

DESIGN
Senior Creative Director
Francisco G Delgadillo
Senior Design Director
Suemi Lam
Design Director
Richard Merchán
Contributing Designers
Jaime Ferrand, Nicholas Pavkovic
Production Designers
Sheila Brennan, Kathy Cygnarowicz

PUBLISHING
Publisher
Jeff Spicer
Production Director and Associate Publisher
Jennifer Hamilton +1.650.506.3794
Senior Manager, Audience Development and Operations
Karin Kinnear +1.650.506.1985

ADVERTISING SALES
Associate Publisher
Kyle Walkenhorst +1.323.340.8585
Northwest and Central U.S.
Tom Cometa +1.510.339.2403
Southwest U.S. and LAD
Shaun Mehr +1.949.923.1660
Northeast U.S. and EMEA/APAC
Mark Makinney +1.805.709.4745
Recruitment Advertising
Tim Matteson +1 310-836-4064
Mailing-List Rentals
Contact your sales representative.

RESOURCES
Oracle Products
+1.800.367.8674 (U.S./Canada)
Oracle Services
+1.888.283.0591 (U.S.)
Oracle Press Books
oraclepressbooks.com

ARTICLE SUBMISSION
If you are interested in submitting an article, please e-mail the editors.

SUBSCRIPTION INFORMATION
Subscriptions are complimentary for qualified individuals who complete the subscription form.

MAGAZINE CUSTOMER SERVICE
java@halldata.com **Phone** +1.847.763.9635

PRIVACY
Oracle Publishing allows sharing of its mailing list with selected third parties. If you prefer that your mailing address or e-mail address not be included in this program, contact Customer Service.

Java Magazine is published bimonthly with a free subscription price by Oracle, 500 Oracle Parkway, MS OPL-3C, Redwood City, CA 94065-1600.

Digital Publishing by Texterity

Java Skills #1 Priority Of Hiring Managers

Get Certified with Oracle University

- ✓ Prepare with Java experts
- ✓ In the classroom or online
- ✓ Pass or retest for free
- ✓ And save up to 20%

Java™ Click to Register

ORACLE®

Source: from Dice.com's *"Dice Report"*; *"January 2012: The Top Spots"*